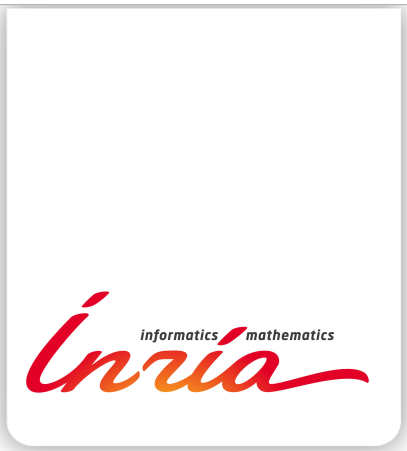
Symbolic Bisimulation for Open and Parameterized Systems - Extended version

Zechen HOU , Eric MADELAINE , Jing LIU , Yuxin DENG

RESEARCH REPORT
N° 9304
November 2019
Project-Team Kairos

ISSN 0249-6399 ISRN INRIA/RR--9304--FR+ENG



Symbolic Bisimulation for Open and Parameterized Systems - Extended version

Zechen HOU *, Eric MADELAINE † ‡, Jing LIU *, Yuxin DENG *

Project-Team Kairos

Research Report n° 9304 — November 2019 — 49 pages

Abstract: *Open Automata* (OA) are symbolic and parameterized models for open concurrent systems. Here *open* means partially specified systems, that can be instantiated or assembled to build bigger systems. An important property for such systems is "compositionality", meaning that logical properties, and equivalences, can be checked locally, and will be preserved by composition. In previous work, a notion of equivalence named *FH-Bisimulation* was defined for open automata, and proved to be a congruence for their composition. But this equivalence was defined for a variant of open automata that are intrinsically infinite, making it unsuitable for algorithmic treatment.
We define a new form of equivalence named *StrFH-Bisimulation*, working on finite encodings of OAs. We prove that StrFH-Bisimulation is consistent and complete with respect to the FH-Bisimulation.
Then we propose two algorithms to check *StrFH-Bisimulation*: the first one requires a (user-defined) relation between the states of two finite OAs, and checks whether it is a StrFH-Bisimulation. The second one takes two finite OAs as input, and builds a "weakest StrFH-bisimulation" such that their initial states are bisimilar. We prove that this algorithm terminates when the data domains are finite. Both algorithms use an SMT-solver as a basis to solve the proof obligations.

Key-words: Concurrent Systems, Symbolic Bisimulation, Open Systems, SMT Solver

KAIROS is a common team between the I3S Lab (Univ. Côte d'Azur and CNRS) and INRIA Sophia Antipolis Méditerranée

* East China Normal University, Shanghai, China
† INRIA Sophia Antipolis Méditerranée, BP 93, 06902 Sophia Antipolis, France
‡ UCA, Université Côte d'Azur

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Bisimulation symbolique pour les systèmes paramétrés ouverts - Version étendue

Résumé : Les *Automates Ouverts* (OA) sont des modèles symboliques et paramétrés pour les systèmes concurrents ouverts. Ici, *open* désigne des systèmes partiellement spécifiés, qui peuvent être instanciés ou assemblés pour construire de plus grands systèmes. Une propriété importante pour de tels systèmes est la "compositionnalité", ce qui signifie que les propriétés logiques et les équivalences peuvent être vérifiées localement et seront préservées par la composition. Dans des travaux antérieurs, une notion d'équivalence nommée *FH-Bisimulation* était définie pour les automates ouverts et se révélait être une congruence pour leur composition. Mais cette équivalence a été définie pour une variante des automates ouverts intrinsèquement infinis, ce qui la rend impropre au traitement algorithmique.

Nous définissons une nouvelle forme d'équivalence nommée *StrFH-Bisimulation*, travaillant sur des encodages finis des OA. Nous prouvons que la StrFH-Bisimulation est cohérente et complète pour la FH-Bisimulation.

Ensuite, nous proposons deux algorithmes pour vérifier la *StrFH-Bisimulation*: le premier nécessite la donnée d'une relation (définie par l'utilisateur) entre les états de deux OA finis, et vérifie s'il s'agit d'une StrFH-Bisimulation. Le second prend deux OA finies en entrée et construit une "StrFH-bisimulation la plus faible" telle que leurs états initiaux soient bisimilaires. Nous prouvons que cet algorithme se termine lorsque les domaines de données sont finis. Les deux algorithmes utilisent un solveur SMT comme base pour résoudre les obligations de preuve.

Mots-clés : Systèmes concurrents, Bisimulation Symbolique, Systèmes ouverts, Solveur SMT

1 Introduction

How to reduce the state space of system models, being a challenging area of formal verification, attracts the interest of many researchers approaching this problem from different angles.

Some approaches focus on the structure of the system models. This is the case with the proposal of open systems, which provide a way to represent incomplete systems, like parallel skeletons, or process algebra operators. The idea is to define (small) open systems for which you can prove crucial properties, then assemble these systems to build full applications, while preserving the properties.

Some approaches are dealing with semantics of the system models in a symbolic way. This provides a way to use abstract (parameterized) terms to describe the states or transitions, manipulating explicitly potentially unbounded data domains, and thus decreasing drastically the model size.

Bisimulation is also an important approach, in which attention on the equivalence between different systems, allows for hierarchical model minimization, enhancing the practical capabilities for further analysis, typically model-checking.

The approach of [1] offers a methodology using open, symbolic, and parameterized models, endowed with a notion of symbolic bisimulation. It defines a new behavioural specification formalism called "Open parameterized Networks of Synchronized Automata (pNet)" for distributed, synchronous, asynchronous or heterogeneous systems. The pNet model has a hierarchical and tree-like structure that gives it a strong ability for describing and composing complex systems. The symbolic and open aspects of pNets give them the potential to use small state space to represent large systems. Figures 1 show examples of pNets inspired from [1], and that we will use as running examples in this paper. We won't go here into the details of the pNet model, because each pNet will generate a corresponding open automaton as it's operational semantics, and all the further analysis and verification are defined on this open automaton.

Current work define open automata with some *semantic* flavor, in which each automaton has an interesting "Closure" property under substitution. These *Semantic Open Automata* are endowed with a notion of symbolic bisimulation called *FH-Bisimulation*, and it is proved in [2] that this equivalence is a congruence for pNet composition, that allows for compositional verification methods. But this definition cannot be used as such in algorithms and tools, because all the semantic open automaton are infinite, according to their closure property.

Another important inspiration for our work was from the seminal research in [3] on Symbolic Transition Graphs with Assignements (STGA). STGA share with open automata the ability to manipulate explicitly symbolic expressions in transitions, guards and assignments. They are endowed with a notion of symbolic bisimulation, and [4] defined an "On-the-fly" algorithm for computing this bisimulation, although this was limited to *data-independent* systems. Also STGAs were addressing only closed systems, so open automata are significantly different.

Our main goal in this work is to define an alternative bisimulation relation, suitable for finitely represented open automata, as can be generated from the semantic rules of open pNets, and to define algorithms checking or generating this relation.

Our main contributions in this paper are the following:

Contribution 1: We propose a new *structural* Bisimulation equivalence called StrFH-Bisimulation between open automata that are not necessarily closed under substitution. This allows us to address the finite systems that can be computed from pNets. We define a correspondence between such "finite" and "semantic" open automata, and prove that StrFH-Bisimulation is correctly and completely corresponding to FH-Bismulation. Thus, we generalize the interesting properties of FH-Bisimulation to StrFH-Bisimulation. Details of this Bisimulation equivalence and its properties are in section 3.

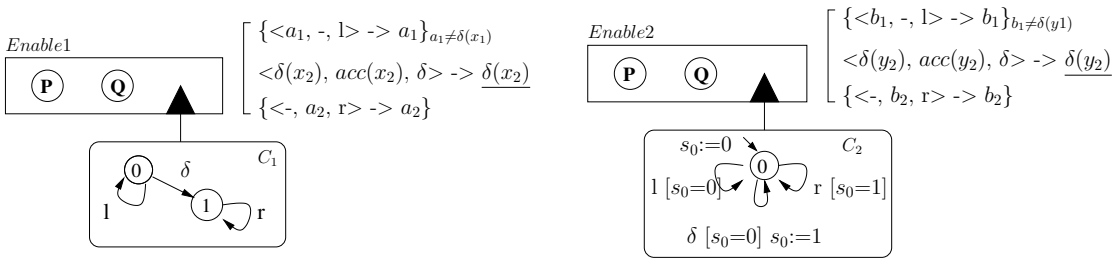


Figure 1: Two pNet encodings for Enable

Contribution 2: The StrFH-Bisimulation we mentioned above, is defined between a kind of open and symbolic systems with parameterized action, value-passing and assignments, all these features will bring great challenges. Our second contribution is proposing an SMT-solver based algorithm that can check if a relation is a StrFH-Bisimulation between input automata. The details of this approach are in Section 4.

Contribution 3: We go further than above approach, and devise a new *on-the-fly* algorithm that generates a weakest StrFH-Bisimulation between two given open automata. We prove that the result of the algorithm is correct and indeed the weakest. The algorithm terminates whenever both the (symbolic) open-automata and the data-domains are finite. Details of this algorithm are in Section 5. Note that this is definitely better than the "data-independence" constraint of the STGA algorithm. If used in association with some abstract interpretation of the data domains, we can address a large set of infinite systems.

2 Background

2.1 Notations

2.1.1 Term algebra

Our models rely on a notion of parameterised actions, that are symbolic expressions using data types and variables. As our model aims at encoding the low-level behaviour of possibly very different programming languages, we do not want to impose one specific algebra for denoting actions, nor any specific communication mechanism. So we leave unspecified the constructors of the algebra that will allow building expressions and actions. Moreover, we use a generic *action interaction* mechanism, based on (some sort of) unification between two or more action expressions, to express various kinds of communication or synchronisation mechanisms.

Formally, we assume the existence of a term algebra $\mathbb{T}$, where Σ is the signature of the data and action constructors. Within $\mathbb{T}$, we distinguish a set of expressions $\mathbb{E}$, including a set of boolean expressions $\mathbb{E}$ ($\mathbb{B} \subseteq \mathbb{E}$). We let e_i range over expressions ($e_i \in \mathbb{E}$). On top of $\mathbb{E}$ we build the action algebra $\mathbb{A}$, with $\mathbb{A} \subseteq \mathbb{T}, \mathbb{E} \cap \mathbb{A} = \emptyset$; naturally action terms will use data expressions as subterms.

Let a range over action labels, op be operators, and x_i range over variable names.

The set of actions is defined as:

$\alpha \in \mathbb{A}$	$::=$	$a(p_1, \ldots, p_n)$	action terms
p_i	$::=$	$?x \mid e_i$	parameters (input variable or expression)
e_i	$::=$	$Value \mid x \mid op(e_1, .., e_n)$	Expressions

The *input variables* in an action term are those marked with the symbol *?*. We additionally assume that each input variable does not appear somewhere else in the same action term: $p_i = ?x \Rightarrow \forall j \neq i.\, x \notin vars(p_j)$

Bound variables are the variables quantified by a quantifier $\forall$ or $\exists$, and other variables are free variables. The function $vars(t)$ identifies the set of free variables in a term $t \in \mathbb{T}$, and $iv(t)$ returns the name of its input variables (without the '?' marker). Action algebras can encode naturally usual point-to-point message passing calculi (using $a(?x_1, ..., ?x_n)$ for inputs, $a(v_1, .., v_n)$ for outputs), but it also allows for more general synchronisation mechanisms, like gate negotiation in Lotos, or broadcast communications.

2.1.2 Substitutions

We denote $(x_k \leftarrow e_k)^{k \in K}$ a substitution, also ranged by σ, where $(x_k)^{k \in K}$ is a set of variables and $(e_k)^{k \in K}$ is a set of expressions. The application of the substitution to an expression is denoted $e'\{\!|(x_k \leftarrow e_k)^{k \in K}|\!\}$, the operation replaces in parallel all free occurrences of the variables $x_k^{k \in K}$ by the expression $e_k^{k \in K}$ in e'. We define $dom(\sigma)$ as the domain of substitution σ, ranged by variables, and $codom(\sigma)$ the set of variables in the right hand side expressions in σ. For example, if $\sigma = (x \leftarrow e_1, y \leftarrow e_2)$ then $dom(\sigma)$ denotes the variable set $\{x, y\}$, and $codom(\sigma)$ is $vars(e_1) \cup vars(e_2)$.

We write $\uplus$ the union operator between substitutions. Suppose σ_1 and σ_2 are two substitutions, then $\{\!|\sigma_1 \uplus \sigma_2|\!\}$ means applying the substitution σ_1 and σ_2 parallel. If there are conflict between σ_1 and σ_2, preference is given to the substitution in σ_1. Formally, suppose a substitution σ_{inter} where $dom(\sigma_{inter}) = dom(\sigma_1) \cap dom(\sigma_2)$ and $\sigma_{inter} \subseteq \sigma_2$, we have: $\{\!|\sigma_1 \uplus \sigma_2|\!\} = \{\!|\sigma_1 \uplus (\sigma_2 \setminus \sigma_{inter})|\!\}$.

We write $\otimes$ the application operator between substitutions, which means apply the second substitution to the first. Formally: $(x_k \leftarrow e_k)^{k \in K} \otimes \sigma = (x_k \leftarrow e_k\{\!|\sigma|\!\})^{k \in K}$.

We write $\odot$ the composition operator between substitutions. When there are two substitution applied sequentially, we can use this operator to compose these two substitutions as one substitution. Formally, $e'\{\!|\sigma_1 \odot \sigma_2|\!\} = (e'\{\!|\sigma_1|\!\})\{\!|\sigma_2|\!\}$

Supposing $\sigma = (x_k \leftarrow e_k)^{k \in K}$, it's easy to see we can derive that:

$$\begin{aligned}(\sigma \otimes \sigma_1) \otimes \sigma_2 &= (x_k \leftarrow e_k\{\!|\sigma_1|\!\})^{k \in K} \otimes \sigma_2 \\ &= (x_k \leftarrow e_k\{\!|\sigma_1|\!\}\{\!|\sigma_2|\!\})^{k \in K} \\ &= (x_k \leftarrow e_k\{\!|\sigma_1 \odot \sigma_2|\!\})^{k \in K} \\ &= \sigma \otimes (\sigma_1 \odot \sigma_2)\end{aligned}$$

2.1.3 Valuation

For a set $V = (x_k)^{k \in K}$ of variables, we denote $\rho(V) = (x_k \leftarrow v_k)^{k \in K}$ a valuation defined on V, where v_k is an element in data domain of x_k. It's easy to see that a valuation is a specific substitution.

For example, let fv be a set of free variables $fv = \{x, y\}$ with x an Integer variable and y a Boolean variable, then $\{x \leftarrow 1, y \leftarrow \mathit{True}\}$ is a valuation defined on fv.

2.2 Open Automaton

Open Automata[2] are semantic symbolic models representing the operational semantics of open data-dependant systems. In[2] a set of structural operational semantics rules is introduced, which defines the behavioural semantics of Open pNets in terms of Open Automata.

Each open automaton consists of a set of *States* and a set of *Open Transitions*, each with its (disjoint) set of *state variables*, and *Open Transitions* relates the behavior of holes (encoding the environment) with the behavior of the system. This section will show the formal definition of an open automaton.

Definition 2.1. ***Open Automaton:*** *An open automaton is a structure $A = \langle J, \mathcal{S}, s_0, \mathcal{T} \rangle$ where:*

- *J is a set of holes indices,*
- *$\mathcal{S}$ is a set of states and s_0 is a state among $\mathcal{S}$,*
- *$\mathcal{T}$ is a set of open transitions and for each $t \in \mathcal{T}$ there exists J' with $J' \subseteq J$, such that t is an open transition over J' and $\mathcal{S}$.*

Definition 2.2. ***Open Transition:*** *An open transition of an open automaton $\langle J, \mathcal{S}, s_0, \mathcal{T} \rangle$ is a structure of the form*

$$\frac{\beta_j^{j \in J'},\ \mathit{Pred},\ \mathit{Post}}{s \xrightarrow{\alpha} s'}$$

where $J' \subseteq J$, $s, s' \in \mathcal{S}$ and β_j is an action of the hole j. Holes (represented above by their indices) are used to represent unspecified subsystems, that are parts of the environment. These unspecified subsystems can have any possible behaviour, but their interaction with the system is specified by the predicate Pred. *α is an action expression denoting the resulting action of this open transition. We call* source variables *of an OT the union of all variables in the terms β_j and α, and the state variables of s. Pred is a predicate over the source variables. Post is a set of variable substitutions represented as $(x_k \leftarrow e_k)^{k \in K}$ where x_k are state variables of s', and e_k are expressions over the source variables. Open transitions are identified modulo logical equivalence on their predicate. In the algorithms, this will be checked using a combination of predicate inclusions.*

Suppose OT is an open transition starting from state s to t. We denote the set of all the free variables in open transition OT as $vars(OT)$, and $vars(s)$ is set of all the state variables of state s. Last, fv_{OT} denotes the set of variables in the open transition OT besides all the state variables. More precisely, $fv_{OT} = vars(OT) \setminus \big(vars(s) \cup vars(s')\big)$.

These definitions will be widely used in subsequent sections.

Figure 2 shows two open automata generated by the pNets in Figure 1. Here we don't need to figure out the process of this generation, and just need to understand that, for any system modeled by a pNet, we can generate an open automaton, representing the behavioral semantics of the original pNet, and any property we proved on this open automaton is also owned by original pNets.

This provides us a clear way to verify the property of systems. We can use some open data-dependent modeling language, like pNet, which are more readable and easier to use, to model

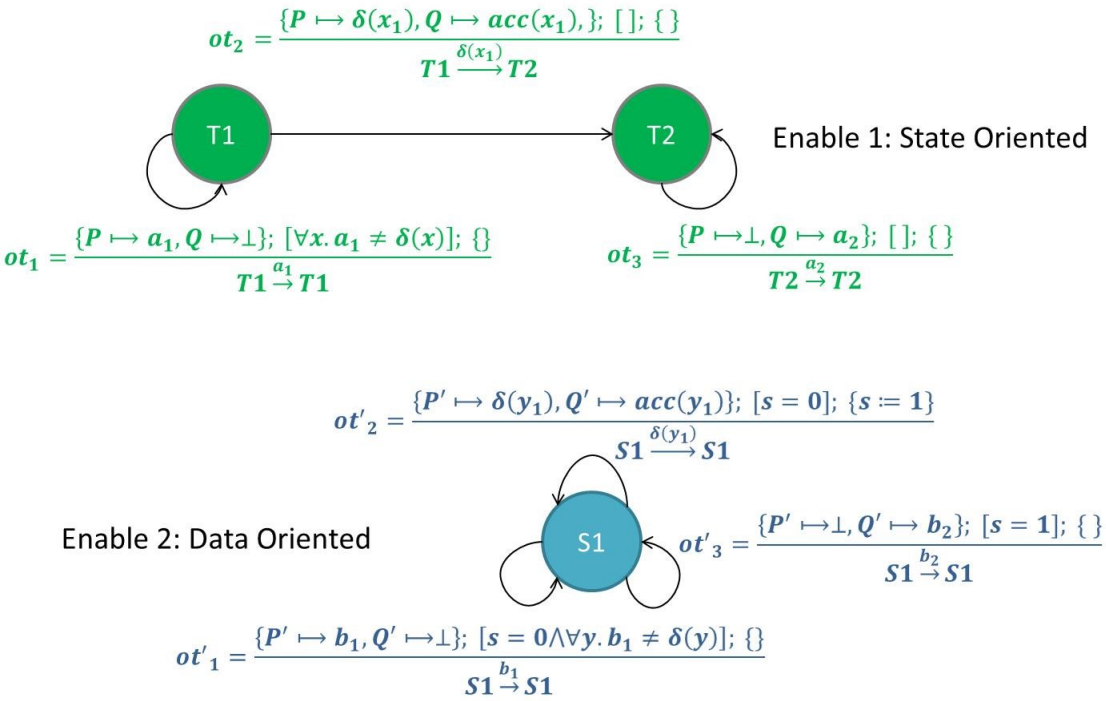


Figure 2: The 2 open automata derived from pNets in Fig. 1

the systems we are interested in. Then, we can use some operational rules to construct an open automaton, which is less readable but easier for operation and verification, from the original system we modeled.

2.3 Semantic Open Automata

Previous researches were taking a semantics and logical understanding of these automata: *Semantic Open Automata* are closed under a simple form of refinement that allows to refine the predicate, or substitute any free variable by an expression. Formally:

Definition 2.3. ***Semantic Open Automata:*** *A semantic open automaton is an open automaton* $oa = \langle J, \mathcal{S}, init, \mathcal{T} \rangle$*, such that for any open transition* $ot \in \mathcal{T}$*:*

Let σ *be any substitution such that* $dom(\sigma) \cap (vars(s) \cup vars(s')) = \emptyset$*, let pred be any predicate on* $vars(ot)$*, then*

$$\frac{\beta\{\sigma\}, Pred\{\sigma\} \wedge pred\{\sigma\},\ Post \otimes \sigma}{s \xrightarrow{\alpha\{\sigma\}} s'} \in \mathcal{T}$$

In fact, the reason why this definition deserves our attention is, it's closely related to the nature of symbolic systems. In semantics, symbolic attribution implies each variables in system can represent a huge set of values. In terms of it, any open transition in an open automaton, can represent large sets of *ground open transitions*, in which all the variables are valuated with constants, as long as these constants satisfy the predicate of original transition. Thus, thanks

to the above "closure" definition, for any open transition OT in a semantic open automata oa, not only OT can represent large sets of ground open transitions, but also for any subset of these ground open transitions, one can always find an open transition OT' in oa, which can represent this subset.

With this definition, researchers defined a relation between semantic open automata, called FH-Bisimulation [2], which have an interesting "Composability" property with respect to pNet composition, allowing compositional reasoning on open systems.

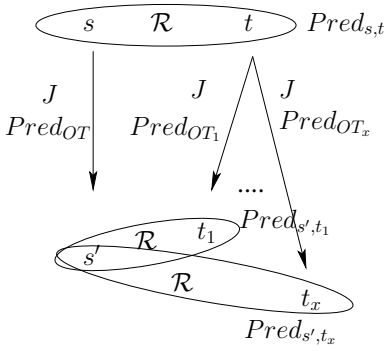
2.4 FH-Bisimulation

Bisimulations are equivalence relations between transition systems, where systems behave in the same way in the sense of one system simulates the other, in terms of the *actions* they do, not of their internal state.

FH-Bisimulation is a kind of strong bisimulation relation between *Semantical Open automata*, introduced in . FH-Bisimulation can be formally defined in the following way:

Definition 2.4. ***FH-Bisimulation:***

Suppose that $A_1 = \langle J, \mathcal{S}_1, init_1, \mathcal{T}_1 \rangle$ *and* $A_2 = \langle J, \mathcal{S}_2, init_2, \mathcal{T}_2 \rangle$ *are two semantic open automata with identical activated holes, with disjoint state variables. Then* $\mathcal{R}$ *is an FH-Bisimulation iff for any triple* $(s, t | Pred_{s,t}) \in \mathcal{R}$ *where* $s \in \mathcal{S}_1 \wedge t \in \mathcal{S}_2$*, we have the following:*



- *For any open transition* $OT \in \mathcal{T}_1$ *starting from* s

$$\frac{\beta_j^{j \in J'},\ Pred_{OT},\ Post_{OT}}{s \xrightarrow{\alpha} s'}$$

there exists a set of open transitions $OT_x^{x \in X} \subseteq \mathcal{T}_2$ *starting from* t

$$\frac{\beta_{j_x}^{j_x \in J'_x},\ Pred_{OT_x},\ Post_{OT_x}}{t \xrightarrow{\alpha_x} t'_x}$$

such that $J' \subseteq J,\ J'_x \subseteq J,\ \forall x. J' = J'_x,\ \{s', t'_x | Pred_{s',t'_x}\} \in \mathcal{R}$ *and*

$$Pred_{s,t} \wedge Pred_{OT} \Longrightarrow \bigvee_{x \in X} (\forall j \in J'. \beta_j = \beta_{j_x} \wedge \alpha = \alpha_{j_x} \wedge Pred_{OT_x} \wedge Pred_{s',t'_x} \{\!| Post_{OT} \uplus Post_{OT_x} |\!\})$$

that means each open transition starting from s *in* $\mathcal{T}_1$ *can be covered by a set of transitions* $OT_x^{x \in X}$ *starting from* t *in* $\mathcal{T}_2$*.*

- *and symmetrically, for any open transition starting from t in $\mathcal{T}_2$, should be covered by a set of transitions starting from s in $\mathcal{T}_1$.*

The word *covered* here means that each *ot* on one side, representing symbolically a set of "ground" transitions, can be covered on the other side by several open transitions on the other sides, each implementing transitions for a subset of *ot* instantiations, and eventually leading to different though still equivalent, states.

The (slightly complicated) implication means that for each instantiation fulfilling $Pred_{s,t} \wedge Pred_{OT}$, it is possible to find an instantiation of one of the corresponding OT_x, satisfying the corresponding property.

It maybe a little unclear also why we need $Pred_{s,t}$ in Triples. In fact, recall the name of FH-Bisimulation, here "FH" is a short cut of "Formal Hypotheses", showing the fact that FH-Bisimulation is a relation based on a certain hypotheses. Thus, $Pred_{s,t}$ in Triples is a formal way to describe this hypotheses, and in particular express the relations between the state variables of the two open automata.

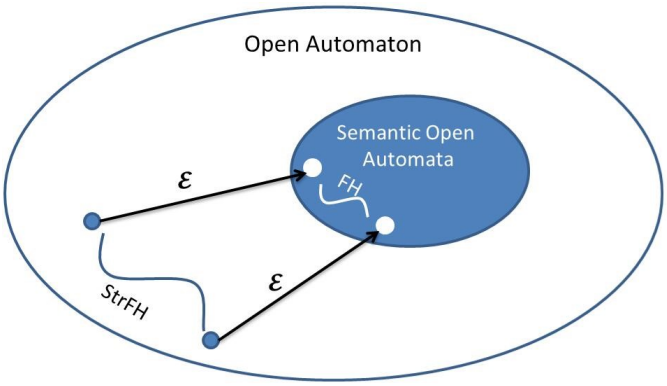
In [2]'s work, it is proved that FH-Bisimulation is an equivalence, which is reflexive, symmetric and transitive. And as mentioned before, FH-Bisimulation also has a powerful property of *Composability*: Given two semantic open automata such that there exist a FH-Bisimulation between them, if we compose these two automata with another semantic open automata (technically, using an open pNet), then the two result automata is still FH-Bisimulation.

And that's an important motivation of our work. Obviously, any semantic open automaton is infinite, thus it's hard to do any verification of FH-Bisimulation on semantic open automata. What we want, is to generalize this FH-Bisimulation to a relation between all the open automata, not only between the "infinite" subset of open automata, and thus it's become possible for us to do any verification on this relation. At next section, we will discuss how we do that.

3 StrFH-Bisimulation

As mentioned before, the *closure* property of semantic open automata bring the important composability properties, but also does not allow implementing FH-Bisimulation algorithms between them. So we want to generalize this FH-Bisimulation to a relation between all the open automata, including the finite ones generated from the pNets semantic rules[2].

In this section, we first give the definition of our new relation *StrFH-Bisimulation*, which is defined between open automata. Then we define an Expansion function $\mathcal{E}$, such that for any open automata we can build a corresponding semantic open automata. Finally, with the above definitions, we will show how our relation is consistent and complete with respect to the FH-Bisimulation. From following figure, we can easily have a first intuition about what all these definitions are:



3.1 StrFH-Bisimulation

We provide a new relation, which is between open automata, called StrFH-Bisimulation, defined formally as:

Definition 3.1. ***StrFH-Bisimulation:*** *Let* $StrA_1 = \langle J, \mathcal{S}_1, init_1, \mathcal{T}_1\rangle$ *and* $StrA_2 = \langle J, \mathcal{S}_2, init_2, \mathcal{T}_2\rangle$ *be two open automata with identical activated holes, and disjoint state variables. Then* $\mathcal{R}$ *is an StrFH-Bisimulation iff for any triple* $(s, t|Pred_{s,t}) \in \mathcal{R}$*, where* $s \in \mathcal{S}_1 \wedge t \in \mathcal{S}_2$ *and we have the following:*

- *For any open transition* $OT \in \mathcal{T}_1$ *starting from* s

$$\frac{\beta_j^{j\in J'},\ Pred_{OT},\ Post_{OT}}{s \xrightarrow{\alpha} s'}$$

there exist a set of open transitions $OT_x^{x\in X} \subseteq \mathcal{T}_2$ *starting from* t

$$\frac{\beta_{j_x}^{j_x\in J'_x},\ Pred_{OT_x},\ Post_{OT_x}}{t \xrightarrow{\alpha_x} t'_x}$$

such that $J'_x \subseteq J,\ \forall x.J' = J'_x,\ \{s', t'_x|Pred_{s',t'_x}\} \in \mathcal{R}$ *and*

$$\forall fv_{OT}.\{Pred_{s,t} \wedge Pred_{OT} \Longrightarrow \bigvee_{x\in X} [\exists fv_{OT_x}.(\forall j \in J'.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_{j_x} \wedge Pred_{OT_x} \wedge Pred_{s',t'_x}\{Post_{OT} \uplus Post_{OT_x}\})]\} \quad (1)$$

that means open transition start from s *in* $\mathcal{T}_1$ *can be covered by a set of transitions* $OT_x^{x\in X}$ *starting from* t *in* $\mathcal{T}_2$

- *and symmetrically, for any open transition starting from* t *in* $\mathcal{T}_2$ *can be covered by a set of transitions starting from* s *in* $\mathcal{T}_1$*.*

There may be another confusing term here: $\forall fv_{OT}.\varphi$, for a set of variables fv_{OT}. This means that for any valuation of all the variables in fv_{OT}, the inside formula is true. For example, for an open transition OT, if $fv_{OT} = \{x, y, z\}$, then $\forall fv_{OT}.\varphi$ means $\forall x, y, z.\varphi$.

3.2 Expansion

Here, we define a Expansion function $\mathcal{E}$, such that for any open automaton, we can use this function to get a corresponding semantic open automaton.

Definition 3.2. ***Expansion:*** *Let OA be the set of all open automata, then we define an Expansion Function $\mathcal{E}$:*

$$\mathcal{E} : OA \to OA$$

where for any open automata $A = \langle J, \mathcal{S}, s_0, \mathcal{T} \rangle$, $\mathcal{E}(A) \in OA$, we can have $\mathcal{E}(A) = \langle J, \mathcal{S}, s_0, \mathcal{T}' \rangle$.

The only difference between A and $\mathcal{E}(A)$ is on transitions set, where $\mathcal{T}'$ is the smallest set of open transitions such that: for any open transition $OT \in \mathcal{T}$

$$\frac{\beta_j^{j\in J},\ Pred_{OT},\ Post_{OT}}{s \xrightarrow{\alpha} s'}$$

let's denote

$$OT\{\!|\sigma, pred|\!\} = \frac{\beta_j^{j\in J}\{\!|\sigma|\!\},\ Pred\{\!|\sigma|\!\} \wedge pred\{\!|\sigma|\!\},\ Post \otimes \sigma}{s \xrightarrow{\alpha\{\!|\sigma|\!\}} s'}$$

Then for any substitution σ where $dom(\sigma) \subseteq fv(OT)$, for any additional predicate $pred$, we have:

$$OT\{\!|\sigma, pred|\!\} \in \mathcal{T}'$$

Obviously, in this definition, after applying function $\mathcal{E}$, the result we have is an open automaton. The following theorem states that this result $\mathcal{E}(A)$ is a semantic open automaton.

Theorem 3.3. *For any open automaton A, the expanded open automaton $\mathcal{E}(A)$ is a semantic open automaton.*

We give a formal proof for this theorem in Appendix A. Briefly, the proof will show that all the open transitions in the open automaton generated by Expansion Function, meet the "Closure" property of Semantic Open Automata.

Note: by construction, the states of $\mathcal{E}(A)$ are same as the states of A. As a consequence, all relations we construct as triple sets $\mathcal{R} = \{(s, t|Pred_{s,t})\}$, are both relations on A and $\mathcal{E}(A)$.

3.3 Correct Correspondence

Here we want to prove that StrFH-Bisimulation correctly corresponds to FH-Bisimulation, as expressed by:

Theorem 3.4. ***Correctness***
For any two open automata A_1 and A_2, construct the two corresponding semantic open automata $\mathcal{E}(A_1)$ and $\mathcal{E}(A_2)$. Then we have: Suppose Triple Set $\mathcal{R}$ is a StrFH-Bisimulation between A_1 and A_2, then $\mathcal{R}$ is also a FH-Bisimulation between $\mathcal{E}(A_1)$ and $\mathcal{E}(A_2)$.

We give a formal proof for Theorem 3.4 in Appendix B.1. Briefly, the intuition of the proof is: for any open transition OT' in $\mathcal{E}(A_1)$, supposing it's original open transition is OT in A_1, we know that OT can be covered with a set of open transition $OT_x^{x\in X}$ in A_2. From this set, We can always construct a set of *ground* open transitions $\{OT'_{x,y}\}^{x\in X \bigwedge y\in Y}$ belonging to $\mathcal{E}(A_2)$ such that this set of open transitions covers the OT'. And symmetrically, for any open transition in $\mathcal{E}(A_2)$.

3.4 Complete Correspondence

Here we want to prove that StrFH-Bisimulation completely corresponds to FH-Bisimulation, as expressed by:

Theorem 3.5. ***Completeness***
For any two open automata A_1 and A_2, construct the two corresponding semantic open automata $\mathcal{E}(A_1)$ and $\mathcal{E}(A_2)$. Then we have: Suppose Triple Set $\mathcal{R}$ is a FH-Bisimulation between $\mathcal{E}(A_1)$ and $\mathcal{E}(A_2)$, then $\mathcal{R}$ is also a StrFH-Bisimulation between A_1 and A_2.

We give a formal proof for Theorem 3.5 in Appendix B.2. The intuition of the proof is: for any open transition OT in A_1, applying any (ground) valuation ρ_i defined on fv_{OT} to OT, the result can be represented by an open transition OT'_i, which belongs to $\mathcal{E}(A_1)$. We know that OT'_i can be covered by a set of open transitions $\{OT'_{x,i}\}^{x\in X}$ in $\mathcal{E}(A_2)$, corresponding to open transitions $\{OT_{x,i}\}^{x\in X}$ in A_2 (may contain duplicate open transition). Collecting all the open transitions $\{OT_{x,y}\}^{x\in X}$ constructed with each valuation ρ_y, and we can finally get a larger set $\{OT_{x,y}\}^{x\in X\wedge y\in Y}$. We will prove OT is covered by this larger set. And symmetrically for any open transition in A_2.

3.5 Running Example

Recall the two open automata in Figure 2, both automata are generated from enable model in Figure 1, respectively by pNets *Enable1* and *Enable2*. It's easy to see, the former automaton, we name it as A_1, uses different states to represent before and after activation (the δ transition); and the latter automaton, we name it as A_2, uses different values of a state variable s to represent that. It's a very typical situation that two equivalent systems essentially express the same meaning in different ways.

Here we provide these two automata as a running example, to illustrate why Triple Set $\{(T_1, S_1|s=0), (T_2, S_1|s=1)\}$ is a StrFH-Bisimulation between these two automata. Details are presented in Appendix B.3.

4 Checking Algorithm

As we mentioned before, in the formal definition, for any two open automata $\langle J_1, \mathcal{S}_1, init_1, \mathcal{T}_1\rangle$, $\langle J_2, \mathcal{S}_2, init_2, \mathcal{T}_2\rangle$, a StrFH-Bisimulation $\mathcal{R}$ is a set of triples $\{(s, t|Pred_{s,t})\}$.

Given two open automata and a set of Triples, it seems that following the definition, we can easily verify if the given relation is a StrFH-Bisimulation. But for that, we need to check whether a first order logic formula is a tautology or not, and due to the fact that first order logic is undecidable, and that we want to use this approach with realistic data types and expressions in the pNets, it may becomes very hard.

Thanks to the development of Satisfiability Modulo Theory (SMT) technology, which has a great ability for solving satisfiable problems of first order logic[5], we have a practical way to do this checking: for any proof obligation, we can generate its negation, and use an SMT-solver to check if it is satisfiable. If yes, thus the original proof obligation is not a tautology. This does not break the undecidability problem, but gives us a semi-decidable method: our StrFH-Bisimulation check is decidable whenever satisfiability of the first-order formulas used in the relation is decidable by our SMT encoding. In practice, this last property depends on the axiomatisation of the data domains and operators used in a particular use case.

In this section we will first explain a basic algorithm, which checks whether a given Triple Set is a StrFH-Bisimulation between two input open automata. Note that, in this algorithm, the user

needs to provide the Triple Set, which may be difficult for some complicated systems. In next section we will give another algorithm which can automatically compute a StrFH-Bisimulation for given open automata.

4.1 Checking Algorithm

algorithm 1 Verify StrFH-Bisimulation with Given Triple Set

input: A_1 and A_2 are two open automata, where $A_1 = \langle J_1, \mathcal{S}_1, init_1, \mathcal{T}_1 \rangle$, $A_2 = \langle J_2, \mathcal{S}_2, init_2, \mathcal{T}_2 \rangle$. $\mathcal{R}$ is a set of triples, where $\mathcal{R} = \{(s, t | Pred_{s,t})\}$.
output: a boolean value, means whether $\mathcal{R}$ is a StrFH-Bisimulation between A_1 and A_2

```
1: function TraverseTriples(A_1, A_2, R)
2:     for each Triple (s, t|Pred_{s,t}) in R do
3:         for each Open Transition OT from state s, suppose it's target is s' do
4:             OTSet ← an empty set for Open Transitions
5:             PredSet ← an empty set for Formula
6:             for each Open Transition OT' from state t, suppose it's target is t' do
7:                 if there exists a Triple (s', t'|Pred_{s',t'}) ∈ R and OT have same activated holes as OT' then
8:                     OTSet.add(OT')
9:                     PredSet.add(Pred_{s',t'})
10:                end if
11:            end for
12:            if OTSet is empty then
13:                return False
14:            end if
15:            res ← Verify(OT, OTSet, Pred_{s,t}, PredSet)
16:            if res is False then
17:                // means proof obligation proofOb is not a tautology.
18:                return False
19:            end if
20:        end for
21:    end for
22:    return True
23: end function
```

From two open automata $A_1 = \langle J_1, \mathcal{S}_1, init_1, \mathcal{T}_1 \rangle$, $A_2 = \langle J_2, \mathcal{S}_2, init_2, \mathcal{T}_2 \rangle$, and a set of triples $\mathcal{R} = \{(s, t | Pred_{s,t})\}$, our algorithm will output whether the given Triple Set is a StrFH-Bisimulation between A_1 and A_2 or not. We explain this algorithm in two parts:

4.1.1 Traverse Triples

The main function is *TraverseTriples*. It this function, we will traverse all the Triples, and check whether all of them meet the definition of StrFH-Bisimulation. For each Triple $(s, t | Pred_{s,t})$, we will check both directions: first, check if all the open transitions from s can be covered by the open transitions from t under the $Pred_{s,t}$; then symmetrically, check if all the open transitions from t can be covered by the open transitions from s under the same predicate. The algorithm is the same for both, thus we only show the first direction.

Let Triple $(s, t|Pred_{s,t})$ be the one we are checking. For each open transition OT from state s, let s' be it's target. Then, we will filter all the open transitions starting from t, and collect a set of open transitions $OTSet$ and a set of formulas $PredSet$, such that for any open transition $OT_x \in OTSet$, we have:

- OT has the same active holes as OT_x
- let t'_x be the target of transition OT_x, then there exists a Triple $(s', t'_x|Pred_{s',t'_x})$ in $\mathcal{R}$

and each predicate formula $Pred_{s',t'_x}$ is the predicate formula of Triple $(s', t'_x|Pred_{s',t'_x})$, corresponding to the open transition OT_x.

Then, we call function *Verify*$(OT, Pred_{s,t}, OTSet, PredSet)$. In that function, will generate and check the proof obligation (that is Formula (1) from Definition 3.1), and return a boolean value as result. If this result is false, means proof obligation is not a tautology, thus Triple $(s, t|Pred_{s,t})$ doesn't meet the definition, and Triple Set $\mathcal{R}$ won't be a StrFH-Bisimulation between A_1 and A_2, so function *TraverseTriples* will return false.

If all the Triples pass this check, it means all the Triples in $\mathcal{R}$ meet the definition, thus we can say $\mathcal{R}$ is a StrFH-Bisimulation between A_1 and A_2 and return true.

4.1.2 Generate and Verify Proof Obligation

Then we come to the details of function *Verify*.

algorithm 2 Generate and Verify Proof Obligation

input: OT is an open transition, $OTSet$ is a set of open transitions, $Pred$ is a formula, $PredSet$ is a set of formulas.

output: *result*, a boolean value, means whether the proof obligation generated from input is a tautology or not.

```
1:  function VERIFY(OT, OTSet, Pred, PredSet)
2:      negateOb ← generate negation of proof obligation
3:      use SMT-solver to check if negateOb is satisfiable
4:      if SMT-solver doesn't terminate then
5:          // negateOb is not decidable by SMT-solver
6:          return FAILED
7:      end if
8:      if negateOb is sat then
9:          // negateOb is satisfiable, means original proof obligation is not a tautology
10:         return False
11:     else
12:         return True
13:     end if
14: end function
```

For the input open transition OT, open transition set $OTSet$, formula $Pred_{s,t}$ and formula set $PredSet$, without losing generality, we can suppose that:

$$OT = \frac{\beta_j^{j \in J'},\ Pred_{OT},\ Post_{OT}}{s \xrightarrow{\alpha} s'}$$

and for any open transition $OT_x \in OTSet$ where $x \in X$ is the index of $OTSet$, we have:

$$OT_x = \frac{\beta_j^{j \in J'_x},\ Pred_{OT_x},\ Post_{OT_x}}{t \xrightarrow{\alpha_x} t'_x}$$

and have a corresponding predicate $Pred_{s',t'_x} \in PredSet$. Thus, the proof obligation by definition as following:

$$\forall fv_{OT}.\{Pred_{s,t} \wedge Pred_{OT} \Longrightarrow \bigvee_{x \in X} [\exists fv_{OT_x}.(\forall j \in J'.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge Pred_{s',t'_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\})]\}$$

and note that if negate this proof obligation, then we will get:

$$\exists fv_{OT}.\{Pred_{s,t} \wedge Pred_{OT} \wedge \bigwedge_{x \in X} [\forall fv_{OT_x}.(\exists j \in J'.\beta_j \neq \beta_{j_x} \vee \alpha \neq \alpha_x \vee \neg Pred_{OT_x} \vee \neg Pred_{s',t'_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\})]\} \quad (2)$$

Thus, we can check the satisfiability of this Formula 2 with SMT-solver, if this Formula 2 is unsatisfiable, means the original proof obligation is a tautology, and function will return True; otherwise, return False.

The checking of proof obligation is strongly depending on SMT-solver, as long as SMT solver can determine all the generated proof obligation, then we can say our algorithm can determine StrFH-Bisimulation problem on given automata and Triple Set. If there exists a proof obligation can't be solved by SMT-solver, our algorithm will return $FAILED$ and terminate.

Thus, our algorithm succeeds whenever the theories (over the data domains, expressions, and predicates) used in the SMT engine are decidable.

4.2 Correctness and Termination

4.2.1 Correctness

The algorithm will traverse all the given triples, check whether all the triples satisfied definition. It's easy to see that, this period is closely corresponding to the definition of StrFH-Bisimulation, so we can say: Given two open automata A_1, A_2 and a set of triples $\mathcal{R}$, then $\mathcal{R}$ is a StrFH-Bisimulation between A_1 and A_2 iff the result of applying A_1, A_2 and $\mathcal{R}$ into this checking algorithm is *True*, when the case is decidable.

4.2.2 Termination

It's easy to see that, for each Triple $(s, t|Pred_{s,t})$, as long as the open transitions starting from s and starting from t are finite, the algorithm will generate finite proof obligations according to this Triple. For each proof obligation, as long as it can be decided by SMT-solver, it will terminate and give a result.

Thus, we can say, as long as we input a finite Triple Set, and each Triple corresponding to finite open transitions(even inputted open automata are infinite), and the case is decidable, our algorithm will terminate.

5 Weakest StrFH-Bisimulation and StrFH-Bisimilar Automata

In the previous section, we gave a checking algorithm, which accepts two open automata and a relation (given as a Triple set), and outputs whether the input Triple set is a StrFH-Bisimulation between the two input open automata.

Obviously, this algorithm has a strong need of a reasonable input. If user failed to find out or correctly encode a proper Triples Set, then algorithm can only get the result that a incorrect Triple set is not a StrFH-Bisimulation, but couldn't answer whether there exists a StrFH-Bisimulation between the two input automata.

And in fact, between any two open automata, there always exists many worthless StrFH-Bisimulation relations. For example, suppose there is a Triple set, where for each Triple $(s, t|Pred_{s,t})$ in this set, $Pred_{s,t}$ is always unsatisfiable. Obviously, this Triple set meet the definition of StrFH-Bisimulation, but it makes no sense.

For these reasons, in this section, we will first give a new algorithm, which can accept two open automata, and output the weakest StrFH-Bisimulation between the given open automata; Then, we will define a new property called StrFH-Bisimilar. If two open automata are StrFH-Bisimilar, it means there exists a meaningful StrFH-Bisimulation between them; From the weakest StrFH-Bisimulation generated, we can check if given open automata are StrFH-Bisimilar. We give a formal proof of our algorithm's correctness, prove a (partial) termination property for this algorithm.

5.1 Generate Weakest StrFH-Bisimulation

We will first give a function called *GenerateWeakestStrFH*. The input of function *GenerateWeakest-StrFH* are two open automata $A_1 = \langle J_1, \mathcal{S}_1, init_1, \mathcal{T}_1 \rangle$ and $A_2 = \langle J_2, \mathcal{S}_2, init_2, \mathcal{T}_2 \rangle$. It's output is a Triple set $TripleSet$, which is a StrFH-Bisimulation between A_1 and A_2. Here we will first describe the procedure of this function, and its pseudo-code in algorithm 3. In this section, we will show the fact that the output is a StrFH-Bisimulation between given open automata. Later in the next section, we will show why it's result is the weakest StrFH-Bisimulation.

First let us give some definitions: We denote (s, t) as a StatePair, where s and t are states from different open automata.

We denote (s_{pre}, t_{pre}) as a pre-StatePair of (s, t) where there exists an open transition from s_{pre} to s, and exists an open transition from t_{pre} to t.

We denote (s_{next}, t_{next}) as a next-StatePair of (s, t) where there exists an open transition from s to s_{next}, and exists an open transition from t to t_{next}.

We denote (s', t') as a reachable StatePair of (s, t) iff there exists a path from (s, t) to (s', t'), where each step in this path is a StatePair move to it's next-StatePair.

At the very beginning, function will construct an empty StatePair stack, named *StateStack*. Function will also construct an empty Triple set, named $TripleSet$.

In fact, here we hope $TripleSet$ become the final output of the function, which means after the function terminating, $TripleSet$ become an StrFH-Bisimulation between A_1 and A_2, and any Triple in $TripleSet$ will meet the requirement: all the proof obligations generated from it are tautologies. And *StateStack* here exists as a register, temporarily cache all StatePairs whose corresponding Triples may not satisfy this requirement.

To initialize *StateStack* and $TripleSet$, function will call a sub-function *PushReachableStatePairs*, inputting two states: $init_1$ and $init_2$ and an empty StatePair stack: *StateStack*. It's a recursive function, and after it finished, *StateStack* will contain all the StatePairs which are reachable from $(init_1, init_2)$. Then, for each StatePair (s, t) in *StateStack*, function will add a corresponding Triple $(s, t|True)$ into a Triple set $TripleSet$.

algorithm 3 Generate Weakest StrFH-Bisimulation

input: A_1, A_2 two open automata, where $A_1 = \langle J_1, \mathcal{S}_1, init_1, \mathcal{T}_1 \rangle$ and $A_2 = \langle J_2, \mathcal{S}_2, init_2, \mathcal{T}_2 \rangle$.
output: $TripleSet$, a Triple's set, which is a Weakest StrFH-Bisimulation between two open automata

```
1: function GenerateWeakestStrFH(StateStack)
2:     StateStack ← an empty stack for StatePair
3:     TripleSet ← an empty set for Triple
4:     PushReachableStatePairs(init_1, init_2, StateStack)
5:     for each StatePair (s, t) in StateStack do
6:         add a Triple (s, t|True) into TripleSet
7:     end for
8:     while StateStack is not empty do
9:         (s, t) ← StateStack.pop()
10:        let i be the index of Triple (s, t|Pred_{s,t}) in the TripleSet
11:        UpdatePredicate(i, TripleSet)
12:        if predicate of Triple (s, t|Pred_{s,t}) has been changed then
13:            for each pre-StatePair (s_pre, t_pre) of (s, t) do
14:                if StateStack doesn't contains this StatePair (s_pre, t_pre) then
15:                    push StatePair (s_pre, t_pre) into StateStack
16:                end if
17:            end for
18:        end if
19:    end while
20: end function
```

After above initialization, function will enter a loop that: pops a StatePairs from *StateStack* and updates the corresponding Triple, loops until the stack *StateStack* becomes empty.

In each loop, let (s, t) be the StatePair just popped, function will find the corresponding Triple $(s, t|Pred_{s,t})$ in the $TripleSet$, and call function *UpdatePredicate* to update this Triple. After processed by function *UpdatePredicate*, Triple $(s, t|Pred_{s,t})$ will definitely meet the requirement: all the proof obligations generated from this Triple are tautology.

Finally, function will check if $(s, t|Pred_{s,t})$ has been updated after calling *UpdatePredicate*. If not, the function will continue to pop next StatePair. If changed, we can't simply go to next StatePair, because the change of $Pred_{s,t}$ may change the proof obligations of other Triples, we need to push their corresponding StatePairs into *StateStack* again.

In more detail, let $\{(s_{pre_x}, t_{pre_x})\}^{x \in X}$ be all pre-StatePairs of (s, t). Due to the fact that $Pred_{s,t}$ has been changed to $newPred_{s,t}$, the proof obligation of Triple $(s_{pre_x}, t_{pre_x}|Pred_{pre_x})$ will also be changed, and we need to update this Triple again. So, we will push StatePair (s_{pre_x}, t_{pre_x}) into *StateStack* , and the function will definitely pop this StatePair and update corresponding Triple at some later time.

We know that, at the beginning, *StateStack* contains all the reachable StatePairs. Each time a StatePair is popped, the corresponding Triple will be update so that it will meet the current requirement. On the other hand, after updating, if this updating changes other Triples' proof obligation, make these Triples may no longer meet the requirement, then their corresponding StatePair will be pushed into stack again. Thus, we can see, if any Triple which does not conclusively meet the requirement, it's corresponding StatePair will be pushed into stack. As long as this function *GenerateWeakestStrFH* terminates, which means the stack becomes empty, all the Triples in $TripleSet$ will meet the requirement, and $TripleSet$ is definitely a StrFH-

Bisimulation between A_1 and A_2.

5.1.1 Sub-Functions

Recall that there are two sub-functions during generating the weakest StrFH-Bisimulation. Here we will give a short summary for these two sub-functions, detailed description and pseudo-code can be seen in Appendix C.1.

Sub-function *PushReachableStatePairs* is a recursive function, a variant of DFS (depth first search). Every time it visits a StatePair (s,t), it will pop (s,t) into stack, and call the same function with next-StatePair of (s,t) recursively. Thus, after function *PushReachableStatePairs* terminates, the stack will contain all the reachable StatePairs from initial input $(init_1, init_2)$.

Sub-function *UpdatePredicate* will generate and verify the proof obligation of given Triples, using the SMT engine, like what we did in Section 4.1.2. If the proof obligation is not a tautology, the function will update the predicate with the conjunction of original predicate and simplified proof obligation. So that after this updating, the new proof obligation generated by this updated predicate will definitely be a tautology.

5.1.2 Weakestness

The output of function *GenerateWeakestStrFH* will be the weakest StrFH-Bisimulation between two given open automata. Remember that for any pair of states (s,t), there can be only one triple $(s,t|Pred_{s,t})$ in a FH-bisimulation relation. Then the weakest StrFH-bisimulation between 2 automata is the one in which each Triple is the weakest:

Definition 5.1. ***Weakest Triple:*** *Triple $(s,t|Pred_{s,t})$ is a weakest Triple between open automata A_1 and A_2, meaning that A_1 and A_2 contains states s and t respectively, and for any predicate formula $Pred_{new}$, which can't imply the predicate $Pred_{s,t}$, any Triple Set containing Triple $(s,t|Pred_{new})$ won't be a StrFH-Bisimulation between A_1 and A_2.*

Theorem 5.2. ***Weakest StrFH-Bisimulation:*** *For a StrFH-Bisimulation $\mathcal{R}$ between open automata A_1 and A_2, all the Triples in $\mathcal{R}$ are Weakest Triples, meaning $\mathcal{R}$ is a Weakest StrFH-Bisimulation.*

Theorem 5.3. ***Weakestness:*** *For any pair of open automata A_1 and A_2, the TripleSet computed by algorithm is the weakest StrFH-Bisimulation between A_1 and A_2.*

We give a detailed proof of these two theorems in Appendix C.2.

Proof of theorem 5.2 relies on the fact that a *weakest* Triple can be either (strictly) logically weaker, or incomparable, to the original Triple, but in both cases having at least one weakest Triple contradicts definition 5.1.

Proof of theorem 5.3 is by mathematical induction over the set of Triplpes: for all Beginning Triples (we define it in proof), we prove that they satisfy Definition 5.1 after updating. Then, let $(s,t|Pred_{s,t})$ be a general Triple, suppose all the other Triple satisfy Definition 5.1, then $(s,t|Pred_{s,t})$ will also satisfy Definition 5.1.

5.2 Checking StrFH-Bisimilar

With above functions, we can finally check if two open automata are StrFH-Bisimilar. First let us give a formal definition for StrFH-Bimilar:

Definition 5.4. ***FH-Bisimilar:*** *Let $A_1 = \langle J_1, \mathcal{S}_1, init_1, \mathcal{T}_1 \rangle$ and $A_2 = \langle J_2, \mathcal{S}_2, init_2, \mathcal{T}_2 \rangle$ be two open automata. A_1 and A_2 are FH-Bisimilar if and only if there exists a StrFH-Bisimulation $\mathcal{R}$ between A_1 and A_2, such that:*

- *for any Triple* $(s, t|Pred_{s,t})$ *in* $\mathcal{R}$*, formula* $(Pred_{s,t} \implies False)$ *is not a tautology, or we can say* $Pred_{s,t}$ *is satisfiable.*
- *there must exists a predicate* $Pred_{init}$ *such that Triple* $(init_1, init_2|Pred_{init}) \in \mathcal{R}$

In this definition, all the predicates are satisfiable, meaning the hypothesis of this Bisimulation are satisfiable; and there exists a initial Triple in this Bisimulation, meaning this Bisimulation will become effective as long as both two open automata start from initial state at same time.

algorithm 4 Check StrFH-Bisimilar

input: A_1, A_2 two open automata, where $A_1 = \langle J_1, \mathcal{S}_1, init_1, \mathcal{T}_1\rangle$ and $A_2 = \langle J_2, \mathcal{S}_2, init_2, \mathcal{T}_2\rangle$.
output: *result*, a boolean value, means whether A_1 and A_2 are StrFH-Bisimilar

```
1: function CheckStrFH-Bisimilar(A_1, A_2)
2:     TripleSet ← GenerateWeakestStrFH(StateStack)
3:     for each Triple (s, t|Pred_{s,t}) ∈ TripleSet do
4:         delete this Triple if Pred_{s,t} is unsatisfiable
5:     end for
6:     if TripleSet contains Triple (init_1, init_2, Pred_{init}) then
7:         return True
8:     else
9:         return False
10:    end if
11: end function
```

After we generated a weakest StrFH-Bisimulation between A_1 and A_2 called $\mathcal{R}$, in which any Triple has weakest predicate, we can easily check if A_1 and A_2 are StrFH-Bisimilar by: first, deleting all the Triples with unsatisfiable Predicate formula, and get a new Triple set $\mathcal{R}_{new}$; then, if $\mathcal{R}_{new}$ contains Triple $(init_1, init_2|Pred_{init})$, we can say A_1 and A_2 are StrFH-Bisimilar. Algorithm 4 shows the pseudo-code of this procedure.

5.3 Algorithm Correctness

Proving this algorithm is correct also contains two parts: *Correctness* and *Completeness*.

Theorem 5.5. ***Correctness:*** *Inputting two open automata* A_1 *and* A_2*, as long as the result of algorithm is* $True$*, means* A_1 *and* A_2 *are StrFH-Bisimilar.*

Theorem 5.6. ***Completeness:*** *Inputting two open automata* A_1 *and* A_2*, if* A_1 *and* A_2 *are StrFH-Bisimilar, then the result of algorithm is* $True$ *as long as the problem is decidable by our algorithm.*

5.3.1 Correctness

We give a formal proof for Theorem 5.5 in Appendix D.1. Key point is: after deleting all the unsatisfiable Triples in the generated StrFH-Bisimulation, the result is still a StrFH-Bisimulation.

5.3.2 Completeness

We give a formal proof for Theorem 5.6 in Appendix D.2. Key point is: Contrapositive of this Theorem will be easier to prove. With the Theorem 5.3, it's easy to see that, as long as the function return $False$, it's impossible to find a StrFH-Bisimulation between given automata which contains a initial Triple $(init_1, init_2|Pred_{init})$ and $Pred_{init}$ is satisfiable.

5.4 Conditional Termination

In this section, we give a conditional termination result for algorithm 4: for any two given open automata, if they are finite, and data domain of all variables in given automata are finite, and the StrFH-Bisimilar problem for these two automata is decidable, then our algorithm will terminate.

Noting that it's may not the most general hypothesis ensuring termination, but it's a very reasonable hypothesis, due to the fact that many widely used realistic systems meet this requirement.

Proof. Suppose there are two finite open automata A_1 and A_2.

Considering the function *GenerateWeakestStrFH*, because A_1 and A_2 are finite, then we will generate a finite $StateStack$, and thus have a finite $TripleSet$.

And for each time we call function *UpdatePredicate* with an input Triple $(s, t|Pred_{s,t})$, due to the fact that both two automata are finite, we will only generate finite proof obligation.

On the other hand, denoting $\{\rho_x\}^{x\in X}$ as set of all possible valuations on all variables. If the data domain of all variables in A_1 and A_2 are finite, we know that $\{\rho_x\}^{x\in X}$ is definitely finite. For any Predicate Formula $Pred_{s,t}$, we denote $\{\rho_y\}^{y\in Y}$ as a set of valuations (defined on all variables, the same below) which can satisfy $Pred_{s,t}$, we know $\{\rho_y\}^{y\in Y}$ is finite, and $Y \subseteq X$. Every times we change the predicate formula $Pred_{s,t}$ to $newPred_{s,t}$ during function *UpdatePredicate*, we know that:

$$newPred_{s,t} = Pred_{s,t} \wedge po$$

where po is the proof obligation. And $Pred_{s,t}$ is being updated implies the fact that there exists some valuations $\rho_{y'}^{y'\in Y}$ which can't satisfy po. Thus, if we denote $\{\rho_z\}^{z\in Z}$ as the set of valuations which can satisfy proof obligation po, it's easy to see that the set of valuations which can satisfy $newPred_{s,t}$ equals to $\{\rho_y\}^{y\in Y} \cap \{\rho_z\}^{z\in Z}$, and it's definitely smaller than $\{\rho_y\}^{y\in Y}$. Thus we come to the conclusion that this kind of change must be finite, because this updating is a decrement within a finite field.

Moreover, for any Triple $(s, t|Pred_{s,t})$, only when there exists at least a new "next-Triple" that has changed predicate, the function *UpdatePredicate* will be called with $(s, t|Pred_{s,t})$ as input.

Due to the fact that next-Triples of any Triple are finite, and the changes of predicates are also finite, we know that there are always finitely many calls to function *UpdatePredicate* with any Triple. Thus, function *GenerateWeakestStrFH* will definitely terminate, and obviously function *CheckStrFH-Bisimilar* will also terminate. □

5.5 Running Example

In Appendix D.3, we also provide the open automata in Fig.2 as a running example for our algorithm, to illustrate it's procedure.

6 Related Work

To the best of our knowledge, there is no other research works on the Bisimulation Equivalence between such complicated (open, symbolic, parameterized, with loops and assignments) system models, with providing fully complete algorithms. For the sake of completeness, we give a brief overview of other Symbolic Bisimulation researches.

One line of research is providing Symbolic Bisimulation for different model or language. In Calder's work [6], they define a symbolic semantic for full LOTOS, with a symbolic bisimulation

over it; Borgstrom et al., Delaune et al. and Buscemi et al. provide symbolic semantic and equivalence for different variants of pi calculus respectively [7, 8, 9], and later in 2012 Liu's work provided symbolic bisimulation for full applied pi calculus [10]. The most recent work, Feng et al. provide a symbolic bisimulation for quantum processes [11]. All the above works are based on models quite different from ours, and most of them have no available implementation.

Another line of research is devising algorithms for computing symbolic Bisimulation Equivalence. In Dovier's work, they provide a rank-based algorithm, layering the input model to compute bisimulation [12]. Baldan et al. also focus on open systems, using a logical programming language *Prolog* to model the systems and compute Bisimulation [13]. Wimmer et al. present a signature-based approach to compute Bisimulation, implemented by using BDDs [14]. Lin [4] presents a symbolic bisimulation between symbolic transition graph with assignments (STGA); as mentioned in the *Introduction*, this work brought us lots of inspiration, but they had a strong "data-independence" constraint that our approach significantly overcomes.

Going further than computing symbolic bisimulation, there are several works on devising approach to minimize the system by symbolic bisimulation. The well-known partition refinement algorithm, which first devised by Paige et al. [15], is unsuitable for symbolic models. Thus Bonchi et al. devised a *symbolic partition refinement* algorithm to compute bisimilarity and redundancy at same time [16]; D'Antoni et al. provided a Forward Bisimulation for minimizing the nondeterministic symbolic finite automata [17]. The above approaches, due to the difference of system models they use, are not applicable to our issues, but we are still inspired a lot from these related work.

7 Conclusion

Based on previous research, we have proposed StrFH-Bisimulation, a new structural Bisimulation equivalence between *open automata* which are symbolic, open and parameterized system models and can address all the finitely represented systems computed from pNets. By proving that StrFH-Bisimulation correctly and completely corresponds to FH-Bisimulation, we generalize the interesting properties of FH-Bisimulation to StrFH-Bisimulation.

After that, we proposed two algorithms for checking and computing StrFH-Bisimulation: the first one requires a (user-defined) relation and two finite open automata, to check whether the given relation is a StrFH-Bisimulation between the given automata; the second one go further, for any two given open automata, it computes the *weakest StrFH-Bisimulation* $\mathcal{R}$ between them, which means all the generated conditions in $\mathcal{R}$ are the (logicaly) weakest. We provided a non-trivial proof to show the "weakestness" of our algorithm, and show that this algorithm terminates when the data domains are finite.

We have started implementing these algorithms, and need to evaluate their practical performances on realistic use cases. There are also several interesting directions to further develop our work. One idea is about computing weakest StrFH-Bisimulation algorithm. Our preliminary termination conditional is certainly not the best we can get. While it is clear that we cannot get unconditional termination, it would be interesting to find a more liberal condition or to improve the algorithm to handle more practical cases.

Second idea is to develop a minimization algorithm from our StrFH-Bisimulation. Minimizing system models is the most common way to use a bisimulation equivalence, especially in the case of hierarchical models like pNets, where minimization can be used in a compositional way.

Noting that we only developed the theory of symbolic bisimulation between open automata for *Strong Bisimulation*. Devising a symbolic bisimulation for *Weak Bisimulaiton*, which takes invisible or internal moves into account, would also be interesting, and some of our colleagues

are already working on that.

References

[1] Henrio, L., Madelaine, E., Zhang, M.: pNets: An expressive model for parameterised networks of processes. In: 2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, IEEE (2015) 492–496

[2] Henrio, L., Madelaine, E., Zhang, M.: A Theory for the Composition of Concurrent Processes. In Albert, E., Lanese, I., eds.: 36th International Conference on Formal Techniques for Distributed Objects, Components, and Systems (FORTE). Volume LNCS-9688 of Formal Techniques for Distributed Objects, Components, and Systems., Heraklion, Greece (June 2016) 175–194

[3] Lin, H.: Symbolic transition graphs with assignment. In: International Conference on Concurrency Theory, Springer (1996) 50–65

[4] Lin, H.: "on-the-fly instantiation" of value-passing processes. In: Formal Description Techniques and Protocol Specification, Testing and Verification. Springer (1998) 215–230

[5] De Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: International conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer (2008) 337–340

[6] Calder, M., Shankland, C.: A symbolic semantics and bisimulation for full LOTOS. In: International Conference on Formal Techniques for Networked and Distributed Systems, Springer (2001) 185–200

[7] Borgström, J., Briais, S., Nestmann, U.: Symbolic bisimulation in the spi calculus. In: International Conference on Concurrency Theory, Springer (2004) 161–176

[8] Delaune, S., Kremer, S., Ryan, M.: Symbolic bisimulation for the applied pi-calculus. In: International Conference on Foundations of Software Technology and Theoretical Computer Science, Springer (2007) 133–145

[9] Buscemi, M.G., Montanari, U.: Open bisimulation for the concurrent constraint pi-calculus. In: European Symposium on Programming, Springer (2008) 254–268

[10] Liu, J., Lin, H.: A complete symbolic bisimulation for full applied pi-calculus. In: International Conference on Current Trends in Theory and Practice of Computer Science, Springer (2010) 552–563

[11] Feng, Y., Deng, Y., Ying, M.: Symbolic bisimulation for quantum processes. ACM Transactions on Computational Logic (TOCL) **15**(2) (2014) 14

[12] Dovier, A., Gentilini, R., Piazza, C., Policriti, A.: Rank-based symbolic bisimulation:(and model checking). Electronic Notes in Theoretical Computer Science **67** (2002) 166–183

[13] Baldan, P., Corradini, A., Ehrig, H., Heckel, R.: Compositional modeling of reactive systems using open nets. In: International Conference on Concurrency Theory, Springer (2001) 502–518

[14] Wimmer, R., Herbstritt, M., Hermanns, H., Strampp, K., Becker, B.: Sigref–a symbolic bisimulation tool box. In: International Symposium on Automated Technology for Verification and Analysis, Springer (2006) 477–492

[15] Paige, R., Tarjan, R.E.: Three partition refinement algorithms. SIAM Journal on Computing **16**(6) (1987) 973–989

[16] Bonchi, F., Montanari, U.: Minimization algorithm for symbolic bisimilarity. In: European Symposium on Programming, Springer (2009) 267–284

[17] D'Antoni, L., Veanes, M.: Forward bisimulations for nondeterministic symbolic finite automata. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer (2017) 518–534

A Expansion Function

Here we give a formal proof for Theorem 3.3, which means the Open Automata generated by Expansion Function is definitely a Semantic Open Automata.

Proof. Let $A = \langle J, \mathcal{S}, s_0, \mathcal{T} \rangle$ be a general open automaton, and $\mathcal{E}(A) = \langle J, \mathcal{S}, s_0, \mathcal{T}' \rangle$ be the open automaton generated from applying function $\mathcal{E}$ on A. Let $OT \in \mathcal{T}'$ be an open transition in $\mathcal{E}(A)$, starting from state s and targeting state s'. By construction there always exists an original open transition OT_{origin} in A:

$$OT_{origin} = \frac{\beta_j^{j\in J},\ Pred_{origin},\ Post_{origin}}{s \xrightarrow{\alpha} s'}$$

together with a substitution σ_{OT} and an additional predicate $pred_{OT}$, such that:

$$OT = OT_{origin}\{\!|\sigma_{OT}, pred_{OT}|\!\}$$

To prove that $\mathcal{E}(A)$ is closed under Definition 3, we need to prove that for any σ_{new} be any substitution where $dom(\sigma_{new}) \subseteq fv(OT)$, and let $pred_{new}$ be any additional predicate on $vars(OT)$, let denote $OT_{new} = OT\{\!|\sigma_{new}, pred_{new}|\!\}$, then $OT_{new} \in \mathcal{T}'$.

Developing the definition, and using substitution properties, we get that :

$$OT_{new} = \frac{\begin{array}{c}\beta_j^{j\in J}\{\!|\sigma_{OT}|\!\}\{\!|\sigma_{new}|\!\},\qquad Pred_{origin}\{\!|\sigma_{OT}|\!\}\{\!|\sigma_{new}|\!\} \wedge pred_{OT}\{\!|\sigma_{OT}|\!\}\{\!|\sigma_{new}|\!\} \\ \wedge pred_{new}\{\!|\sigma_{new}|\!\},\qquad (Post_{origin} \otimes \sigma_{OT}) \otimes \sigma_{new}\end{array}}{s \xrightarrow{\alpha\{\!|\sigma_{OT}|\!\}\{\!|\sigma_{new}|\!\}} s'}$$

$$= \frac{\begin{array}{c}\beta_j^{j\in J}\{\!|\sigma_{OT} \odot \sigma_{new}|\!\},\qquad Pred_{origin}\{\!|\sigma_{OT} \odot \sigma_{new}|\!\} \\ \wedge\big(pred_{OT}\{\!|\sigma_{OT}|\!\} \wedge pred_{new}\big)\{\!|\sigma_{new}|\!\},\qquad Post_{origin} \otimes (\sigma_{OT} \odot \sigma_{new})\end{array}}{s \xrightarrow{\alpha\otimes(\sigma_{OT}\odot\sigma_{new})} s'}$$

Now let Δ be a new (boolean) variable which didn't occurred in open transition OT_{origin} and $pred_{OT}$, such that

$$\Delta \notin \big(vars(\beta_j) \cap vars(Pred_{origin}) \cap vars(pred_{OT}) \cap vars(Post_{origin}) \cap vars(\alpha)\big)$$

and construct a new substitution $\sigma_{extra} = (\Delta \leftarrow pred_{new})$. Thus, we can derive OT like following:

$$OT_{new} = \frac{\begin{array}{c}\beta_j^{j\in J}\{\!|(\sigma_{OT} \odot \sigma_{new}) \uplus \sigma_{extra}|\!\},\qquad Pred_{origin}\{\!|(\sigma_{OT} \odot \sigma_{new}) \uplus \sigma_{extra}|\!\} \\ \wedge\big(pred_{OT} \wedge \Delta\big)\{\!|(\sigma_{OT} \odot \sigma_{new}) \uplus \sigma_{extra}|\!\},\qquad Post_{origin} \otimes [(\sigma_{OT} \odot \sigma_{new}) \uplus \sigma_{extra}]\end{array}}{s \xrightarrow{\alpha\{\!|(\sigma_{OT}\odot\sigma_{new})\uplus\sigma_{extra}|\!\}} s'}$$

thus, we have constructed a substitution $\sigma_{final} = (\sigma_{OT} \odot \sigma_{new}) \uplus \sigma_{extra}$, and a additional predicate $pred_{final} = pred_{OT} \wedge \Delta$, such that

$OT_{new} = OT_{orig}\{\!|\sigma_{final}, pred_{final}|\!\}$, so $OT_{new} \in \mathcal{T}'$. □

B Correspondence between StrFH-Bisimulation and FH-Bisimulation

B.1 Proof of Correctness

In this section, we will prove that StrFH-Bisimulation correctly corresponds to FH-Bisimulation, which is formally expressed by Theorem 3.4. Before attempting to prove this theorem, we start with several Lemma which may help the proof.

Lemma B.1. *Let A_1, A_2 be two open automata, OT a transition of A_1, and $OT_x^{x\in X}$ a set of transitions of A_2:*

$$OT = \frac{\beta_j^{j\in J'},\ Pred_{OT},\ Post_{OT}}{s \xrightarrow{\alpha} s'}$$

$$OT_x = \frac{\beta_j^{j\in J'_x},\ Pred_{OT_x},\ Post_{OT_x}}{t \xrightarrow{\alpha_x} t'_x}$$

Then the following formula

$$\forall fv_{OT}.\{Pred_{s,t} \wedge Pred_{OT} \implies \bigvee_{x\in X} [\exists fv_{OT_x}.(\forall j.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge Pred_{s',t'_x}\{\!|Post_{OT} \uplus Post_{OT_x}|\!\})]\} \quad (3)$$

holds if and only if for any valuation $\rho_y(fv_{OT})$ defined on fv_{OT}, there exist a set of valuations $\{\rho_{x,y}(fv_{OT_x})\}^{x\in X}$ defined on fv_{OT_x} respectively, such that the following formula holds:

$$\Big\{Pred_{s,t} \wedge Pred_{OT} \implies \bigvee_{x\in X} \Big[\Big(\forall j.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge Pred_{s',t'_x}\{\!|Post_{OT} \uplus Post_{OT_x}|\!\}\Big)\{\!|\rho_{x,y}(fv_{OT_x})|\!\}\Big]\Big\}\{\!|\rho_y(fv_{OT})|\!\} \quad (4)$$

Generally speaking, in above Lemma, we hope to convert a first-order logic formula consisting of quantifiers, into a formula with quantified valuations. Intuitively, forall quantifier $\forall$ means *for any valuation, the formula will hold*; while existence quantifier $\exists$ means *there exist a valuation such that the formula will hold.*

Lemma B.1 use this intuitive sense, rewriting the Formula 3. Of cause, intuition is not enough for proving a Lemma, we give a formal proof as following:

Proof. First we prove the Necessity of Lemma B.1, that is (3) => (4). In the given situation, suppose formula 3 holds. Let $\rho_y(fv_{OT})$ be any valuation defined on fv_{OT}, applying $\rho_y(fv_{OT})$ to formula (3) we will get:

$$\Big\{Pred_{s,t} \wedge Pred_{OT} \implies \bigvee_{x\in X} [\exists fv_{OT_x}.(\forall j.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge Pred_{s',t'_x}\{\!|Post_{OT} \uplus Post_{OT_x}|\!\})]\Big\}\{\!|\rho_y(fv_{OT})|\!\} \quad (5)$$

Considering the existential quantifier in formula 5, for any variable $a_i \in fv_{OT_x}$, let $C_{x,i}$ be the existent value for variable a_i. Then we can construct a set of valuation $\{\rho_{x,y}(fv_{OT_x})\}^{x\in X}$ such that for any $a_i \in fv_{OT_x}$, we have a valuation $(a_i \leftarrow C_{x,i})$ in $\rho_{x,y}(fv_{OT_x})$. Obviously, with this constructed set of valuation, formula 4 will holds.

For any valuation $\rho_y(fv_{OT})$ there exist a valuation set $\{\rho_{x,y}(fv_{OT_x})\}^{x\in X}$ such that formula 4 holds, thus the Necessity is proved.

Next we prove the Sufficiency of Lemma B.1, that is (4) => (3). In the given situation, suppose that for any valuation $\rho_y(fv_{OT})$ defined on fv_{OT}, there exist a set of valuations $\{\rho_{x,y}(fv_{OT_x})\}^{x\in X}$, such that formula 4 holds.

Considering that making valuation set $\{\rho_{x,y}(fv_{OT_x})\}^{x\in X}$ effective in formula 4, all the variables in $\{fv_{OT_x}\}^{x\in X}$ will be valuated with a constant value, which means all these variables has an extent constant value can make formula 4 hold. This implies the fact that following formula

$$\Big\{Pred_{s,t} \wedge Pred_{OT} \implies \bigvee_{x\in X} \big[\exists fv_{OT_x}.(\forall j.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge Pred_{s',t'_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\})\big]\Big\}\{\!\{\rho_y(fv_{OT})\}\!\} \quad (6)$$

holds.

On the other hand, for any variables $a_i \in fv_{OT}$ assigned with any constant value C_i, we can always construct a corresponding valuation $\rho'(fv_{OT})$ such that for any $a_i \in fv_{OT}$, have a valuation $(a_i \leftarrow C_i)$ in $\rho'(fv_{OT})$. With the constructed valuation $\rho'(fv_{OT})$, we know following formula

$$\Big\{Pred_{s,t} \wedge Pred_{OT} \implies \bigvee_{x\in X} \big[\exists fv_{OT_x}.(\forall j.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge Pred_{s',t'_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\})\big]\Big\}\{\!\{\rho'(fv_{OT})\}\!\} \quad (7)$$

holds. Imaging that if we make this constructed valuation $\rho'(fv_{OT})$ effective in formula 7, any variable $a_i \in fv_{OT}$ will be valuated with C_i, and formula 7 still holds.

For any variables in fv_{OT} assigned with any value, the formula 7 always holds, it's meaning is definitely same as the ForAll quantifier $\forall fv_{OT}$. With using this quantifier, we will have formula 3, and it holds. Thus Sufficiency is proved. □

Lemma B.2. *For any formula φ, any substitution σ, if φ holds then $\varphi\{\!\{\sigma\}\!\}$ holds. Formally, we have*

$$\varphi \Longrightarrow \varphi\{\!\{\sigma\}\!\}$$

Moreover, considering that valuation is a kind of substitution, for any valuations ρ, we have

$$\varphi \Longrightarrow \varphi\{\!\{\rho\}\!\}$$

Lemma B.2 is a basic property for logic language with substitutions and valuations, so we won't give a formal proof for it.

With the above lemmas, we can now prove Theorem 3.4.

Proof. Let A_1, A_2 be two open automata $A_1 =< J, \mathcal{S}_1, init_1, \mathcal{T}_1 >$ and $A_2 =< J, \mathcal{S}_2, init_2, \mathcal{T}_2 >$, and their expansions $\mathcal{E}(A_1) =< J, \mathcal{S}_1, init_1, \mathcal{T'}_1 >$ and $\mathcal{E}(A_2) =< J, \mathcal{S}_2, init_2, \mathcal{T'}_2 >$.

Then suppose there exists a Triple set $\mathcal{R}$ that is an StrFH-Bisimulation between A_1 and A_2. We want to prove that $\mathcal{R}$ is also a FH-Bisimulation between $\mathcal{E}(A_1)$ and $\mathcal{E}(A_1)$.

Let $(s, t|Pred_{s,t})$ be any triple in $\mathcal{R}$. By construction, for any open transition $OT' \in \mathcal{T'}_1$ starting from state s in $\mathcal{E}(A_1)$, there exists at least one origin open transition $OT \in \mathcal{T}_1$ starting from state s in A_1 :

$$OT = \frac{\beta_j^{j\in J'},\ Pred_{OT},\ Post_{OT}}{s \xrightarrow{\alpha} s'}$$

and there exists a substitution σ and an additional predicate $pred$, such that

$$OT' = \frac{\beta_j^{j\in J'}\{\!\!\{\sigma\}\!\!\},\ Pred_{OT}\{\!\!\{\sigma\}\!\!\} \wedge pred\{\!\!\{\sigma\}\!\!\},\ Post_{OT} \otimes \sigma}{s \xrightarrow{\alpha\{\!\!\{\sigma\}\!\!\}} s'}$$

Due to the fact that $\mathcal{R}$ is a StrFH-Bisimulation between A_1 and A_2, and $(s, t|Pred_{s,t})$ is a Triple in $\mathcal{R}$, we know that there exists a set of open transitions $OT_x^{x\in X} \subseteq \mathcal{T}_2$ starting from state t in A_2:

$$OT_x = \frac{\beta_{j_x}^{j_x\in J'_x},\ Pred_{OT_x},\ Post_{OT_x}}{t \xrightarrow{\alpha_x} t'_x}$$

from Lemma B.1, suppose that $\{\rho_y\}^{y\in Y}$ contains all possible valuations defined on variable set fv_{OT}, we have that for any valuation $\rho_y^{y\in Y}(fv_{OT})$, there exists a set of valuations $\{\rho_{x,y}(fv_{OT_x})\}^{x\in X}$ which corresponding to open transitions OT_x respectively, such that the following formula

$$\Big\{ Pred_{s,t} \wedge Pred_{OT} \Longrightarrow \bigvee_{x\in X} \Big[\Big(\forall j.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge Pred_{s',t'_x}\{\!\!\{Post_{OT} \uplus Post_{OT_x}\}\!\!\} \Big) \{\!\!\{\rho_{x,y}(fv_{OT_x})\}\!\!\} \Big] \Big\} \{\!\!\{\rho_y(fv_{OT})\}\!\!\}$$

holds.

For any valuation $\rho_y^{y\in Y}(fv_{OT})$ and any open transition $OT_x^{x\in X}$ with corresponding valuation $\rho_{x,y}(fv_{OT_x})$, we can construct an open transition $OT'_{x,y}$:

$$OT'_{x,y} = \frac{\beta_{j_x}^{j_x\in J'_x}\{\!\!\{\rho_{x,y}(fv_{OT_x})\}\!\!\},\qquad Pred_{OT_x}\{\!\!\{\rho_{x,y}(fv_{OT_x})\}\!\!\},\ Post_{OT_x} \otimes \rho_{x,y}(fv_{OT_x})}{t \xrightarrow{\alpha_x \rho_{x,y}(fv_{OT_x})} t'_x}$$

it's easy to see that, $OT'_{x,y}$ is an open transition in $\mathcal{E}(A_2)$, with substitution $\sigma = \rho_{x,y}(fv_{OT_x})$ and additional predicate $pred_2 = True$. From this rule, we will construct a large set of open transitions $\{OT'_{x,y}\}^{x\in X\wedge y\in Y}$. Before continuing to prove, we would like to have a new lemma first, to help the rest part of proof:

Lemma B.3. *With all the above pre-condition, we have following*

$$Pred_{s,t} \wedge Pred_{OT} \wedge AddPred_{OT} \Longrightarrow \bigvee_{x\in X}\bigvee_{y\in Y}\Big[(\forall j.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge Pred_{s',t'_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\})\{\!\{\rho_{x,y}(fv_{OT_x})\}\!\}\Big] \quad (8)$$

holds.

Proof. Here we assume that Lemma B.3 doesn't hold, and try to reduction to absurdity. From formula 8, after unfolding the *Implies*, we have:

$$\neg Pred_{s,t} \vee \neg Pred_{OT} \vee \neg AddPred_{OT} \vee \bigvee_{x\in X}\bigvee_{y\in Y}\Big[(\forall j.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge Pred_{s',t'_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\})\{\!\{\rho_{x,y}(fv_{OT_x})\}\!\}\Big]$$

and the negation of above formula is the following:

$$Pred_{s,t} \wedge Pred_{OT} \wedge AddPred_{OT} \wedge \bigwedge_{x\in X}\bigwedge_{y\in Y}\neg\Big[(\forall j.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge Pred_{s',t'_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\})\{\!\{\rho_{x,y}(fv_{OT_x})\}\!\}\Big] \quad (9)$$

Suppose the Lemma B.3 doesn't hold, means that with the given pre-condition, formula 8 doesn't hold. Thus, the negation of formula 9 is satisfiable, and we can say that there exist a set of valuation ρ where $dom(\rho)$ equals all the free variables in formula 9, such that

$$\Big\{Pred_{s,t} \wedge Pred_{OT} \wedge AddPred_{OT} \wedge \bigwedge_{x\in X}\bigwedge_{y\in Y}\neg\Big[(\forall j.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge Pred_{s',t'_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\})\{\!\{\rho_{x,y}(fv_{OT_x})\}\!\}\Big]\Big\}\{\!\{\rho\}\!\} \quad (10)$$

holds. We define V as the set of all the free variables in formula 9, due to the fact that all the free variables in formula 9 comes from open transitions OT and $OT_x^{x\in X}$, and $vars(OT) = fv_{OT} \cup (vars(s) \cup vars(s'))$, it's easy to see that

$$V = vars(OT) \bigcup_{x\in X} vars(OT_x) = fv_{OT} \cup vars(s) \cup vars(s') \bigcup_{x\in X}(fv_{OT_x} \cup vars(t) \cup vars(t'_x))$$

All the variables in fv_{OT_x} have been valuated with a concrete value by valuation $\rho_{x,y}(fv_{OT_x})$ in formula 10, thus for any variable in fv_{OT_x}, ρ has no effect on them, so ρ doesn't need to contain

any valuation defined on variables in fv_{OT_x}. We know that fv_{OT} is disjoint with the set of state variables involved in these transitions. Let $vars(s, s', t, t'_x)$ denote the set of variables: $vars(s) \cup vars(s') \cup vars(t) \cup_{x \in X} vars(t'_x)$. Then we can see, there must exist a valuation $\rho_{neg}(fv_{OT})$ defined on fv_{OT}, and a valuation $\rho_{neg}(vars(s, s', t, t'_x))$, such that $\rho = \rho_{neg}(fv_{OT}) \cup \rho_{neg}(vars(s, s', t, t'_x))$ and by unfolding ρ on formula 10, we have:

$$\Big\{ Pred_{s,t} \wedge Pred_{OT} \wedge AddPred_{OT} \wedge \bigwedge_{x \in X} \bigwedge_{y \in Y} \neg \Big[\big(\forall j. \beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge Pred_{s',t'_x} \{\!\{ Post_{OT} \uplus Post_{OT_x} \}\!\} \big) \{\!\{ \rho_{x,y}(fv_{OT_x}) \}\!\} \Big] \Big\} \{\!\{ \rho_{neg}(fv_{OT}) \}\!\} \{\!\{ \rho_{neg}(vars(s, s', t, t'_x)) \}\!\} \quad (11)$$

holds.

Now let's consider the pre-condition. From Lemma B.1, due to the fact that $neg \in Y$, we know that for this negative valuation $\rho_{neg}(fv_{OT})$, there also exists a set of valuations $\rho_{x,neg}(fv_{OT_x})^{x \in X}$ that make the following

$$\Big\{ Pred_{s,t} \wedge Pred_{OT} \Longrightarrow \bigvee_{x \in X} \Big[\Big(\forall j. \beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge Pred_{s',t'_x} \{\!\{ Post_{OT} \uplus Post_{OT_x} \}\!\} \Big) \{\!\{ \rho_{x,neg}(fv_{OT_x}) \}\!\} \Big] \Big\} \{\!\{ \rho_{neg}(fv_{OT}) \}\!\}$$

hold. From Lemma B.2, we know that applying this last valuation $\rho_{neg}(vars(s, s', t, t'_x))$ to the above formula, we will have following

$$\Big\{ Pred_{s,t} \wedge Pred_{OT} \Longrightarrow \bigvee_{x \in X} \Big[\Big(\forall j. \beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge Pred_{s',t'_x} \{\!\{ Post_{OT} \uplus Post_{OT_x} \}\!\} \Big) \{\!\{ \rho_{x,neg}(fv_{OT_x}) \}\!\} \Big] \Big\} \{\!\{ \rho_{neg}(fv_{OT}) \}\!\} \{\!\{ \rho_{neg}(vars(s, s', t, t'_x)) \}\!\}$$

still holds. But due to the fact that $neg \in Y$, this formula is obviously conflict with formula 11, so we have proved Lemma B.3 holds. □

From Lemma B.3, we know that formula 8 holds. And from Lemma B.2, we know that after

applying substitution σ on the formula 8,

$$\Big\{ Pred_{s,t} \wedge Pred_{OT} \wedge AddPred_{OT} \Longrightarrow \bigvee_{x \in X} \bigvee_{y \in Y} \Big[(\forall j.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge Pred_{s',t'_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\})\{\!\{\rho_{x,y}(fv_{OT_x})\}\!\} \Big] \Big\}\{\!\{\sigma\}\!\}$$

still holds. Due to the fact that A_1 and A_2 have disjoint variables, we can distribute the substitutions into the formula and get following

$$\begin{aligned} Pred_{s,t} \wedge Pred_{OT}\{\!\{\sigma\}\!\} \wedge AddPred\{\!\{\sigma\}\!\} \Longrightarrow \\ \bigvee_{x \in X} \bigvee_{y \in Y} (\forall j.\beta_j\{\!\{\sigma\}\!\} = \beta_{j_x}\{\!\{\rho_{x,y}(fv_{OT_x})\}\!\} \\ \wedge \alpha\{\!\{\sigma\}\!\} = \alpha_x\{\!\{\rho_{x,y}(fv_{OT_x})\}\!\} \wedge Pred_{OT_x}\{\!\{\rho_{x,y}(fv_{OT_x})\}\!\} \\ \wedge Pred_{s',t'_x}\{\!\{(Post_{OT} \otimes \sigma) \uplus (Post_{OT_x} \otimes \rho_{x,y}(fv_{OT_x}))\}\!\}) \end{aligned} \quad (12)$$

holds. Note that the above formula is the same as the proof obligation in the definition of FH-Bisimulation, which means OT' can be covered by the open transition set $\{OT'_{x,y}\}^{x \in X \wedge y \in Y}$.

In summary, above proof has proves that: for any triples $(s, t|Pred_{s,t})$ in a StrFH-Bisimulation $\mathcal{R}$ between A_1 and A_2, and for any open transition OT' start from s in $\mathcal{E}(A_1)$, one can always find a large set of open transitions $OT_{x,y}^{x \in X \wedge y \in Y}$ starting from t in $\mathcal{E}(A_2)$, such that formula (12) holds. And symmetrically, for any open transition OT" starting from t in $\mathcal{E}(A_2)$ the same property holds.

This result means, the StrFH-Bisimulation $\mathcal{R}$ is also a FH-Bisimulation between $\mathcal{E}(A_1)$ and $\mathcal{E}(A_2)$. □

B.2 Proof of Completeness

In this section, we will prove that StrFH-Bisimulation is completely corresponding to FH-Bisimulation, which is formally expressed by Theorem 3.5.

Proof. Suppose there are two general open automata $A_1 = \langle J_1, \mathcal{S}_1, init_1, \mathcal{T}_1 \rangle$ and $A_2 = \langle J_2, \mathcal{S}_2, init_2, \mathcal{T}_2 \rangle$, and we generate two semantic open automata $\mathcal{E}(A_1) = \langle J_1, \mathcal{S}_1, init_1, \mathcal{T}'_1 \rangle$ and $\mathcal{E}(A_2) = \langle J_2, \mathcal{S}_2, init_2, \mathcal{T}'_2 \rangle$ through expansion.

Then suppose there exists a triple set $\mathcal{R}$ that is a FH-Bisimulation between $\mathcal{E}(A_1)$ and $\mathcal{E}(A_2)$. What we want to prove is: triple set $\mathcal{R}$ is also a StrFH-Bisimulation between A_1 and A_2.

For any triple $(s, t|Pred_{s,t}) \in \mathcal{R}$, let $OT \in \mathcal{T}_1$ be a general open transition starting from s in A_1

$$OT = \frac{\beta_j^{j \in J'},\ Pred_{OT},\ Post_{OT}}{s \xrightarrow{\alpha} s'}$$

then for any valuation $\rho_y(fv_{OT})$ defined on variable set fv_{OT}, we can construct an open transition OT' with substitution $\sigma = \rho_y(fv_{OT})$ and additional predicate $addPred = True$:

$$OT' = \frac{\beta_j^{j \in J'}\{\!|\rho_y(fv_{OT})|\!\},\ Pred_{OT}\{\!|\rho_y(fv_{OT})|\!\} \wedge True,\qquad Post_{OT} \otimes \rho_y(fv_{OT})}{s \xrightarrow{\alpha\{\!|\rho_y(fv_{OT})|\!\}} s'}$$

Obviously, OT' is an open transition in $\mathcal{E}(A_1)$. Due to the fact that $\mathcal{E}(A_1)$ and $\mathcal{E}(A_2)$ are FH-Bisimilar, we know that there exist a set of open transitions $\{OT'_x\}^{x \in X}$ in $\mathcal{E}(A_2)$, which start from state t. For each OT'_x in this set, there must exist an origin open transition OT_x in A_2

$$OT_x = \frac{\beta_{j_x}^{j_x \in J'_x},\ Pred_{OT_x},\ Post_{OT_x}}{t \xrightarrow{\alpha_x} t'}$$

and there exist a substitution σ_x and a additional predicate $addPred_x$, such that $OT'_x = OT_x\{\!|\sigma_x|\!\}$, and through definition of FH-Bisimulation, we know that following formula

$$\begin{aligned} Pred_{s,t} \wedge Pred_{OT}\{\!|\rho_y(fv_{OT})|\!\} \wedge True \Longrightarrow \\ \bigvee_{x \in X} \{\forall j.\beta_j\{\!|\rho_y(fv_{OT})|\!\} = \beta_{j_x}\{\!|\sigma_x|\!\} \\ \wedge \alpha\{\!|\rho_y(fv_{OT})|\!\} = \alpha_x\{\!|\sigma_x|\!\} \wedge Pred_{OT_x}\{\!|\sigma_x|\!\} \wedge addPred_x\{\!|\sigma_x|\!\} \\ \wedge Pred_{s',t'_x}\{\!|(Post_{OT} \otimes \rho_y(fv_{OT})) \uplus (Post_{OT_x} \otimes \sigma_x)|\!\}\} \end{aligned}$$

holds. Then using the following properties mentioned in previous sections:

$$\begin{gathered} vars(OT) \cap vars(OT_x) = \emptyset \\ dom(\sigma_x) \subseteq fv_{OT_x},\ dom(\rho_y(fv_{OT})) = fv_{OT} \\ (vars(Pred_{s,t}) \cup vars(Pred_{s',t'_x})) \cap (fv_{OT} \cup fv_{OT_x}) = \emptyset \end{gathered}$$

we can extract two substitutions out of the bracket and have the following formula

$$\Big\{ Pred_{s,t} \wedge Pred_{OT} \wedge True \Longrightarrow \bigvee_{x \in X} \Big[(\forall j.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge addPred_x \wedge Pred_{s',t'_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\})\{\!\{\sigma_x\}\!\}\Big]\Big\}\{\!\{\rho_y(fv_{OT})\}\!\} \quad (13)$$

holds. From the basic property of first order logic

$$\varphi \Longrightarrow \exists x, y, z.\varphi$$

we can have the following

$$\{\forall j.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge addPred_x \wedge Pred_{s',t'_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\}\} \Longrightarrow \exists fv_{OT_x}.\{\forall j.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge Pred_{s',t'_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\}\}$$

we can use another way to describe above formula: due to the existential quantifier, we know there always exists a valuation $\rho_{x,y}(fv_{OT_x})$ on variables fv_{OT_x}, such that following

$$\{\forall j.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge addPred_x \wedge Pred_{s',t'_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\}\} \Longrightarrow \{\forall j.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge Pred_{s',t'_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\}\}\{\!\{\rho_{x,y}(fv_{OT_x})\}\!\}$$

holds. From the Lemma B.2, applying σ_x, we have that there exist a valuation $\rho_{x,y}(fv_{OT_x})$ such that

$$\{\forall j.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge addPred_x \wedge Pred_{s',t'_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\}\}\{\!\{\sigma_x\}\!\} \Longrightarrow \{(\forall j.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge Pred_{s',t'_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\})\{\!\{\rho_{x,y}(fv_{OT_x})\}\!\}\}\{\!\{\sigma_x\}\!\}$$

holds. We know that $dom(\sigma_x) \subseteq fv_{OT_x}$ while $dom(\rho_y(fv_{OT_x})) = fv_{OT_x}$, thus after applying $\rho_y(fv_{OT_x})$, all the variables in fv_{OT_x} will be substituted with a constant value, which means σ_x will have no effect on right part of above formula, remove this substitution won't change the

formula. So we can have the following: there exists a valuation $\rho_{x,y}(fv_{OT_x})$ such that

$$\begin{aligned}\big\{\forall j.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \\ \wedge addPred_x \wedge Pred_{s',t'_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\}\{\!\{\sigma_x\}\!\} \\ \Longrightarrow \\ \big\{\big(\forall j.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \\ \wedge Pred_{s',t'_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\}\big)\{\!\{\rho_{x,y}(fv_{OT_x})\}\!\}\big\}\end{aligned}$$

holds. Let's expand this property to all the open transitions $OT_x^{'x\in X}$, we can have that: there exists a set of valuations $\{\rho_{x,y}(fv_{OT_x})\}^{x\in X}$, where each $\rho_{x,y}(fv_{OT_x})$ corresponding to OT'_x respectively, we have following

$$\begin{aligned}\bigvee_{x\in X}\big\{\big(\forall j.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge addPred_x \\ \wedge Pred_{s',t'_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\}\big)\{\!\{\sigma_x\}\!\}\big\} \\ \Longrightarrow \\ \bigvee_{x\in X}\big\{\big(\forall j.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \\ \wedge Pred_{s',t'_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\}\big)\{\!\{\rho_{x,y}(fv_{OT_x})\}\!\}\big\} \quad (14)\end{aligned}$$

holds. Finally, applying formula 14 to formula 13, we get: for any valuation $\rho_y(fv_{OT})$ on fv_{OT}, there always exists a set of valuations $\{\rho_{x,y}(fv_{OT_x})\}^{x\in X}$ which correspond to $OT_x^{'x\in X}$ respectively, such that

$$\begin{aligned}\Big\{Pred_{s,t} \wedge Pred_{OT} \wedge True \Longrightarrow \\ \bigvee_{x\in X}\Big[\big(\forall j.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge addPred_x \\ \wedge Pred_{s',t'_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\}\big)\{\!\{\rho_{x,y}(fv_{OT})\}\!\}\Big]\Big\}\{\!\{\rho_y(fv_{OT})\}\!\}\end{aligned}$$

holds. Note that $\rho_y(fv_{OT})$ is a general valuation defined on fv_{OT}. Thus, from Lemma B.1, we know that it can be deducted that

$$\begin{aligned}\forall fv_{OT}.\Big\{Pred_{s,t} \wedge Pred_{OT} \Longrightarrow \\ \bigvee_{x\in X}\big[\exists fv_{OT_x}.(\forall j.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge Pred_{s',t'_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\})\big]\Big\} \quad (15)\end{aligned}$$

holds, which is definitely the proof obligation of StrFH-Bisimulation.

In summary, above proof has proved that: for any triple $(s,t|Pred_{s,t})$ in a FH-Bisimulation $\mathcal{R}$ between $\mathcal{E}(A_1)$ and $\mathcal{E}(A_2)$, selecting any open transition OT starting from state s in A_1, one can always find a set of open transitions $\{OT_x\}^{x\in X}$ starting from state t in A_2, such that formula 15 holds. And obviously, selecting any open transition $OT^{''}$ starting from state t in A_2 also share same property.

This result means, a FH-Bisimulation $\mathcal{R}$ between $\mathcal{E}(A_1)$ and $\mathcal{E}(A_2)$ is also a StrFH-Bisimulation between A_1 and A_2. Thus, using a StrFH-Bisimulation to represent a corresponding FH-Bisimulation is complete. □

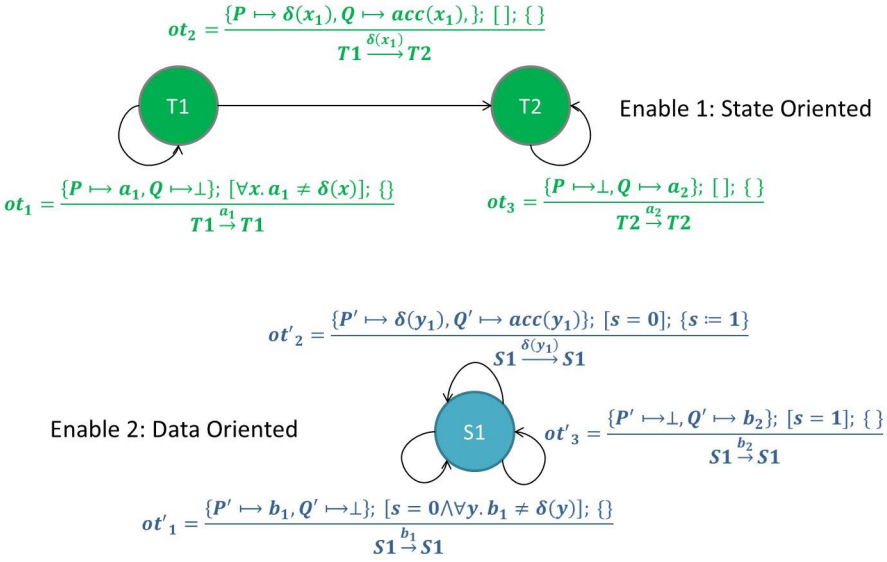


Figure 3: The two open automata

B.3 Running Example for StrFH-Bisimulation

In this section, we will provide the open automata in figure above as a running example, and show why Triple Set $\{(T_1, S_1|s = 0), (T_2, S_1|s = 1)\}$ is a StrFH-Bisimulation between these two automata. We name the first automaton as A_1 and second one as A_2.

Starting from the first Triple $(T_1, S_1|s = 0)$ in given Triple set, it's easy to see that there are five open transitions starting from T_1 or S_1, thus we will construct five proof obligations from this Triple.

For any open transition OT with corresponding set of open transitions $OT_x^{x \in X}$, We know the form of proof obligation is

$$\forall fv_{OT}.\{Pred_{s,t} \wedge Pred_{OT} \Longrightarrow \bigvee_{x \in X} [\exists fv_{OT_x}.(\forall j \in J'.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_{j_x} \wedge Pred_{OT_x} \wedge Pred_{s',t'_x}\{\!|Post_{OT} \uplus Post_{OT_x}|\!\})]\}$$

Thus for open transition ot_1 starting from T_1, with corresponding open transition set $\{ot'_1\}$(all the other open transitions from S_1 have different activated holes), we can build proof obligation following:

$$\forall a_1.\Big\{s = 0 \wedge \forall x.[a_1 \neq \delta(x)] \implies \exists b_1.\{a_1 = b_1 \wedge a_1 = b_1 \wedge s = 0 \wedge \forall y.[b_1 \neq \delta(y)] \wedge s = 0\}\Big\}$$

and it's a tautology, means ot_1 can be covered by ot'_1 under the hypothesis $s = 0$. Then for open

transition ot_2 with corresponding open transition set $\{ot'_2\}$, we can build following:

$$\forall x_1.\{s = 0 \wedge True \implies \exists y_1.[\delta(x_1) = \delta(y_1) \wedge acc(x_1) = acc(y_1) \wedge \delta(x_1) = acc(y_1) \wedge s = 0 \wedge (s = 1\{\!|s \leftarrow 1|\!\})]\}$$

again it's a tautology, means ot_2 can be covered by ot'_2 under the hypothesis $s = 0$. The proof obligations for ot'_1 and for ot'_2 are similar with the above, thus we will not show them.

The last proof obligation of Triple $(T_1, S_1|s = 0)$ may be more interesting. It is built from ot'_3, with corresponding open transition set $\emptyset$:

$$\forall b_2.\{s = 0 \wedge s = 1 \implies False\}$$

because for all the open transitions starting from T_1, none have same activated holes with ot'_3, the formula on right side of "Implication" operator are unsatisfiable, represented by $False$. Despite this, the proof obligation is still a tautology, because it can be derived as $False \implies False$. That means, when Automaton A_1 is in state T_1 and Automaton A_2 is in state S_1, this open transition ot'_3 won't be activated with the given hypothesis $s = 0$. Thus, we know that all the proof obligations built from Triple $(T_1, S_1|s = 0)$ are tautology.

From Triple $(T_2, S_1|s = 1)$, we can build three proof obligations. The procedures of building them are very close to the above, even simpler, so we will leave it to the readers. And it's easy to see all the proof obligation from Triple $(T_2, S_1|s = 1)$ are also tautologies, thus Triple set $\{(T_1, S_1|s = 0), (T_2, S_1|s = 1)\}$ meets the definition of StrFH-Bisimulation, and we can say it's a StrFH-Bisimulation between A_1 and A_2.

C Generate Weakest StrFH-Bisimulation

C.1 Detail of sub-functions

Push Reachable StatePairs

First let us focus on a sub-function *PushReachableStatePairs*, which will be called during Initialization period in *GenerateWeakestStrFH*. It's input are two States s and t belonging to two different open automata, and a StatePair's stack called *StateStack*.

The function is a variant of DFS (Depth First Search). It first visit StatePair (s,t) and push it into the stack *StateStack*, then for each next-StatePair (s_{next_x}, t_{next_x}) of (s,t), call the function *PushReachableStatePairs* with inputting this next-StatePair recursively. Note that each time visiting a StatePair (s,t), function will record that this StatePair has been visited, and won't visit it again.

It's easy to see that, all the StatePair which are reachable from (s,t), will be visited during this recursion. Thus obviously, after end of this recursive function, for any StatePair (s,t) in *StateStack*, all the StatePairs which are reachable from (s,t) will also exist in *StateStack*. Algorithm 5 is the pseudo-code for this recursive function.

algorithm 5 Push Reachable StatePairs

input: s, t two states, where s belong to A_1 and t belong to A_2. *StateStack* a StatePair stack.

```
1: function PushReachableStatePairs(s, t, StateStack)
2:     if StatePair (s, t) has been visited then
3:         return
4:     end if
5:     push StatePair (s, t) into StateStack
6:     mark that (s, t) has been visited
7:     for each open transition OT starting from s and targeting to s'_x do
8:         for each open transition OT' starting from t and targeting to t'_y do
9:             PushReachableStatePairs(s'_x, t'_y, StateStack)
10:        end for
11:    end for
12: end function
```

Update Predicate

Then, let us focus on another sub-function *UpdatePredicate*, which will also be called by *GenerateWeakestStrFH*. It's input is $TripleSet$, a Triple set, and i, index of an element in $TripleSet$. What function *UpdatePredicate* do is updating the Triple $TripleSet[i]$, to make sure that this Triple can meet the Definition of a Triple in StrFH-Bisimulation. Algorithm 6 is the pseudo-code of this function.

More detailed, suppose Triple $TripleSet[i] = (s, t|Pred_{s,t})$, then for any open transition OT starting from s

$$OT = \frac{\beta_j^{j\in J'},\ Pred_{OT},\ Post_{OT}}{s \xrightarrow{\alpha} s'}$$

and let set $\{OT_x\}^{x\in X}$ be all open transitions starting from state t and have the same active holes

than OT

$$OT_x = \frac{\beta_j^{j\in J'_x},\ Pred_{OT_x},\ Post_{OT_x}}{t \xrightarrow{\alpha_x} t'_x}$$

With $\{OT_x\}^{x\in X}$, we can construct a proof obligation po

$$\forall fv_{OT}.\{Pred_{s,t} \wedge Pred_{OT} \implies \bigvee_{x\in X} [\exists fv_{OT_x}.(\forall j.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge Pred_{s',t'_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\})]\}$$

Then we use SMT solver to check the negation of proof obligation po, same as what we did in checking algorithm in last section. If po is a tautology, function will keep going on; If not, means we need to update the Triple's Predicate formula.

Without losing generality, suppose it will be update to $(s, t|newPred_{s,t})$. We have $newPred_{s,t} = Pred_{s,t} \wedge po$, where po is the proof obligation just generated. Then we have $newPred_{s,t}$:

$$Pred_{s,t} \wedge \forall fv_{OT}.\{Pred_{s,t} \wedge Pred_{OT} \implies \bigvee_{x\in X} [\exists fv_{OT_x}.(\forall j.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge Pred_{s',t'_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\})]\}$$

and due to the fact that $vars(Pred_{s,t}) \cap fv_{OT} = \emptyset$, we can distribute $Pred_{s,t}$ into the quantifier and derive $newPred_{s,t}$ to:

$$Pred_{s,t} \wedge \forall fv_{OT}.\Big\{Pred_{s,t} \bigwedge \{Pred_{s,t} \wedge Pred_{OT} \implies \bigvee_{x\in X} [\exists fv_{OT_x}.(\forall j.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge Pred_{s',t'_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\})]\}\Big\}$$

and then simplify it to:

$$Pred_{s,t} \wedge \forall fv_{OT}.\Big\{Pred_{OT} \implies \bigvee_{x\in X} [\exists fv_{OT_x}.(\forall j.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge Pred_{s',t'_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\})]\Big\}$$

It's easy to see that, in the above formula, all the variables in fv_{OT} and fv_{OT_x} are bound variables, and thus free variables in above formula represented by $vars(newPred_{s,t}) = vars(s) \cup vars(t) \cup_{x\in X} vars(Pred_{s',t'_x})$, only contains state variables, meet the definition of the Predicate in a Triple.

Moreover, if we use C to represent following formula:

$$\bigvee_{x\in X} [\exists fv_{OT_x}.(\forall j.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge Pred_{s',t'_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\})]$$

then we can use following to represent predicate formula $newPred_{s,t}$:

$$Pred_{s,t} \wedge \forall fv_{OT}.\{Pred_{OT} \implies C\}$$

and we can generate new proof obligation *newpo* from above new Predicate

$$\forall fv_{OT}.\{newPred_{s,t} \wedge Pred_{OT} \implies C\}$$

and we can derive *newpo* to :

$$\forall fv_{OT}.\{[Pred_{s,t} \wedge \forall fv_{OT}.(Pred_{OT} \implies C) \wedge Pred_{OT}] \implies C\}$$

which is obviously a tautology. Thus we can say, after updating any Triple, all the proof obligation constructed by this Triple is a tautology.

algorithm 6 UpdatePredicate

1: **function** UPDATEPREDICATE($i, TripleSet$)
2: $(s, t|Pred_{s,t}) \leftarrow TripleSet[i]$
3: **for** each open transition OT starting from s to s' **do**
4: using the same way as *Checking Algorithm*, to collect an open transition set $\{OT_x\}^{x\in X}$ from t to t'_x and a corresponding predicate formula set $\{Pred_{s',t'_x}\}^{x\in X}$
5: **if** $\{OT_x\}^{x\in X}$ is an empty set **then**
6: $set[i] \leftarrow (s, t|False)$
7: **return**
8: **end if**
9: use OT, $\{OT_x\}^{x\in X}$, $Pred_{s,t}$ and $\{Pred_{s',t'_x}\}^{x\in X}$ to generate proof obligation po
10: **if** $negate(po)$ is unSatisfiable **then**
11: $Pred_{s,t} \leftarrow Pred_{s,t} \wedge po$
12: **end if**
13: **end for**
14: **for** each open transition OT' starting from t to t' **do**
15: using the same way as *Checking Algorithm*, to collect an open transition set $\{OT'_y\}^{y\in Y}$ from s to s'_y and a corresponding predicate formula set $\{Pred_{s'_y,t'}\}^{y\in Y}$
16: **if** $\{OT'_y\}^{y\in Y}$ is an empty set **then**
17: $set[i] \leftarrow (s, t|False)$
18: **return**
19: **end if**
20: use OT', $\{OT'_y\}^{y\in Y}$, $Pred_{s,t}$ and $\{Pred_{s'_y,t'}\}^{y\in Y}$ to generate proof obligation po
21: **if** $negate(po)$ is unSatisfiable **then**
22: $Pred_{s,t} \leftarrow Pred_{s,t} \wedge po$
23: **end if**
24: **end for**
25: **if** $Pred_{s,t}$ is unSatisfiable **then**
26: $Pred_{s,t} \leftarrow False$
27: **end if**
28: $set[i] \leftarrow (s, t|Pred_{s,t})$
29: **end function**

C.2 Proof of Weakestness

In this section, we will prove that the StrFH-Bisimulation generated by function *GenerateWeakest-StrFH* is the weakest, which is formally expressed by Theorems 5.2 and 5.3.

We start with Theorem 5.2:

Proof. Suppose $\mathcal{R}$ is a StrFH-Bisimulation between A_1 and A_2, and all the Triples in $\mathcal{R}$ are weakest according to Definition 5.1. Let's assume there exists a weaker StrFH-Bisimulation $\mathcal{R}_{weak}$, which means two possibilities:

- either there exists a Triple $(s, t|Pred_{weaker})$ in $\mathcal{R}_{weaker}$, and $Pred_{weaker}$ is weaker than the predicate in the corresponding Triple in $\mathcal{R}$.
- or there exists a Triple $(s, t|Pred_{incomp}$ in $\mathcal{R}_{weaker}$, and $Pred_{incomp}$ is incomparable with the predicate in the corresponding Triple in $\mathcal{R}$.

It's easy to see that, both situations violate the definition of Weakest Triple, thus $\mathcal{R}$ is the Weakest StrFH-Bisimulation. □

Now we prove Theorem 5.3 that states that each Triple in the result of algorithm *GenerateWeakestStrFH* is, individually, a weakest Triple.

Proof. Now we use Mathematical induction to prove Theorem 5.3. First, during calling function *GenerateWeakestStrFH*, after initialize the $TripleSet$, we will definitely face a kind of Triples $(s, t|Pred_{s,t})$, where:

- it's predicate formula $Pred_{s,t}$ has initial value *True*;
- and for any next-Triple $(s_{next_x}, t_{next_x}|Pred_{next_x})$ of $(s, t|Pred_{s,t})$ (meaning there exists an open transition from s to s_{next_x} and an open transition from t to t_{next_x}), $Pred_{next_x}$ has also initial value *True*.

and we call them *Beginning Triples*.

First, let Triple $(s, t|True)$ be any Beginning Triple. For any open transition OT starting from s:

$$OT = \frac{\beta_j^{j\in J'},\ Pred_{OT},\ Post_{OT}}{s \xrightarrow{\alpha} s'}$$

and set of open transitions $\{OT_x\}^{x\in X}$ starting from t

$$OT_x = \frac{\beta_j^{j\in J'_x},\ Pred_{OT_x},\ Post_{OT_x}}{t \xrightarrow{\alpha_x} t'_x}$$

we will have proof obligation po_{OT}:

$$\forall fv_{OT}.\{True \wedge Pred_{OT} \Longrightarrow \bigvee_{x\in X} [\exists fv_{OT_x}.(\forall j \in J'.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge True)]\}$$

and if po_{OT} is a tautology, we don't need to update the initial Predicate *True*, it's the weakest; if po_{OT} is not a tautology, according to function *UpdatePredicate*, we will generate a new predicate formula $newPred_{OT}$. To simplify the formula, we use term C_x to represent following formula:

$$\forall j \in J'.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x}$$

thus, we can represent the updated predicate formula $newPred_{OT}$ as:

$$True \wedge \forall fv_{OT}.\{Pred_{OT} \Longrightarrow \bigvee_{x \in X} [\exists fv_{OT_x}.(C_x \wedge True)]\}$$

Suppose predicate $pred_{weak}$ is any predicate formula, that there exists a StrFH-Bisimulation $\mathcal{R}$ between given automata and this StrFH-Bisimulation contains the Triple $(s, t|Pred_{weak})$.

We know that $(s, t|Pred_{s,t})$ is a *Begining Triples*, which means the predicate of any next-Triple $(s_{next_x}, t_{next_x}|Pred_{next_x})$ is *True* and couldn't be weaker. Thus, the proof obligation generated from $Pred_{weak}$

$$\forall fv_{OT}.\{Pred_{weak} \wedge Pred_{OT} \Longrightarrow \bigvee_{x \in X} [\exists fv_{OT_x}.(C_x \wedge True)]\}$$

should be a tautology.

Then, from the basic rule of propositional logic, we can have:

$$\forall fv_{OT}.\Big\{Pred_{weak} \implies \Big\{Pred_{OT} \Longrightarrow \bigvee_{x \in X} [\exists fv_{OT_x}.(C_x \wedge True)]\Big\}\Big\}$$

and note that $Pred_{weak}$ is a predicate formula in Triple, which only contains state variables as free variables, means $vars(Pred_{weak}) \cap fv_{OT} = \emptyset$, thus we have:

$$Pred_{weak} \implies \forall fv_{OT}.\{Pred_{OT} \Longrightarrow \bigvee_{x \in X} [\exists fv_{OT_x}.(C_x \wedge True)]\}$$

and it can be derive to:

$$Pred_{weak} \implies newPred_{OT}$$

thus we can say, for any predicate formula $Pred_{weak}$, which can make the proof obligation generated from it be a tautology, definitely implies the updated predicate formula $newPred_{OT}$.

Obviously, base on the above conclusion, we can know: for any predicate formula $Pred'$, which can't imply predicate $newPred_{OT}$, there always exists a proof obligation generated by Triple $(s, t|Pred')$ is not a tautology, and any Triple Set containing $(s, t|Pred')$ won't be a StrFH-Bisimulation between given automata. Thus Triple $(s, t|newPred_{OT})$ satisfies the Definition 5.1.

For each proof obligation $po_y^{y \in Y}$ constructed from $(s, t|Pred_{s,t})$, $newPred_y$ is the updated predicate formula from our algorithm, and we know that Triple $(s, t|newPred_y)$ satisfies the Theorem 5.1. Considering all the proof obligations constructed from Triple $(s, t|Pred_{s,t})$, we want all of them be tautologies. Thus, it's easy to see that Triple $(s, t|\bigwedge_{y \in Y} newPred_y)$ also satisfies the Theorem 5.1.

Note that function *UpdatePredicate* will update original predicate to the conjunction of $newPred_y^{y \in Y}$, So, the final predicate $newPred_{s,t} = \bigwedge_{y \in Y} newPred_y$ generated from function satisfies the Theorem 5.1.

In summary, we can say, after applying any Beginning Triple $(s,t|True)$ to function *UpdatePredicate*, the result Triple will satisfy the Theorem 5.1.

Then, we can focus on other Triples. Without losing generality, let Triple $(s,t|Pred_{s,t})$ be any Triple which isn't Beginning Triple and need to be updated, and suppose all the Triples already satisfied Theorem 5.1. For any open transition OT starting from s:

$$OT = \frac{\beta_j^{j\in J'},\ Pred_{OT},\ Post_{OT}}{s \xrightarrow{\alpha} s'}$$

and set of open transitions $\{OT_x\}^{x\in X}$ starting from t

$$OT_x = \frac{\beta_j^{j\in J'_x},\ Pred_{OT_x},\ Post_{OT_x}}{t \xrightarrow{\alpha_x} t'_x}$$

we will have proof obligation po_{OT}:

$$\forall fv_{OT}.\{Pred_{s,t} \wedge Pred_{OT} \Longrightarrow \bigvee_{x\in X} [\exists fv_{OT_x}.(\forall j \in J'.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x} \wedge Pred_{s',t'_x}\{\!|Post_{OT} \uplus Post_{OT_x}|\!\})]\}$$

Same as before, to simplify the formula, we use C_x to represent

$$\forall j \in J'.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \wedge Pred_{OT_x}$$

and then, with using function *UpdatePredicate*, we may update the predicate to $newPred_{OT}$, and we can represent it as following:

$$Pred_{s,t} \wedge \forall fv_{OT}.\{Pred_{OT} \Longrightarrow \bigvee_{x\in X} [\exists fv_{OT_x}.(C_x \wedge Pred_{s',t'_x}\{\!|Post_{OT} \uplus Post_{OT_x}|\!\})]\} \quad (16)$$

Again, suppose formula $Pred_{weak}$ is any predicate formula, that there exists a StrFH-Bisimulation between given automata such that this StrFH-Bisimulation contains Triple $(s,t|Pred_{weak})$.

We know that all the Triples satisfy the Definition 5.1, which means the predicate of any the next-Triple $(s',t'_x|Pred_{s',t'_x})$ should be implied in the proof obligation.

Thus, the proof obligation generated from predicate $Pred_{weak}$

$$\forall fv_{OT}.\{Pred_{weak} \wedge Pred_{OT} \Longrightarrow \bigvee_{x\in X} [\exists fv_{OT_x}.(C_x \wedge Pred_{s',t'_x}\{\!|Post_{OT} \uplus Post_{OT_x}|\!\})]\}$$

should be a tautology, and we can have

$$Pred_{weak} \Longrightarrow \forall fv_{OT}.\{Pred_{OT} \Longrightarrow \bigvee_{x\in X} [\exists fv_{OT_x}.(C_x \wedge Pred_{s',t'_x}\{\!|Post_{OT} \uplus Post_{OT_x}|\!\})]\}$$

same as before.

Then we need to prove $Pred_{weak}$ implies $newPred_{OT}$. Note that Triple $(s,t|Pred_{s,t})$ is not a Beginning Triple, means two possibilities:

- $Pred_{s,t}$ is initial value *True*, but there exist at least one next-Triple $(s', t'|Pred')$ of it, which has just been updated after the beginning, thus $Pred'$ is not initial value *True*.
- $Pred_{s,t}$ is not initial value *True*, but the fact that we are calling function *UpdatePredicate* to update this Triple, means there exists at least one next-Triple $(s', t'|Pred')$ of it, which has just been updated after the last update of Triple $(s, t|Pred_{s,t})$.

For the first situation, we can derive $newPred_{OT}$ to

$$\forall fv_{OT}.\{Pred_{OT} \Longrightarrow \bigvee_{x \in X} [\exists fv_{OT_x}.(C_x \wedge Pred_{s',t'_x}\{\!\!\{Post_{OT} \uplus Post_{OT_x}\}\!\!\})]\}$$

and have

$$Pred_{weak} \implies newPred_{OT}$$

And for the second situation, we can collect a set of predicate formula: $\{Pred_{s',t'_y}\}^{y \in Y} \subseteq \{Pred_{s',t'_x}\}^{x \in X}$, where for any predicate formula $Pred_{s',t'_y}$, it has just been updated after the last update of Triple $(s, t|Pred_{s,t})$, and we can represent it as $Pred_{s',t'_y} = oldPred_{s',t'_y} \wedge newPred_{s',t'_y}$, where $oldPred_{s',t'_y}$ is the predicate formula before last update of Triple $(s', t'_y|Pred_{s',t'_y})$, and $newPred_{s',t'_y}$ is the additional predicate formula during updating.

Then, we can derive that

$$\begin{aligned} Pred_{weak} \implies \forall fv_{OT}.\big\{Pred_{OT} \Longrightarrow & \bigvee_{z \in X \setminus Y} [\exists fv_{OT_z}.(C_z \wedge Pred_{s',t'_z}\{\!\!\{Post_{OT} \uplus Post_{OT_z}\}\!\!\})] \\ & \bigvee_{y \in Y} \big[\exists fv_{OT_y}.\big(C_y \wedge (oldPred_{s',t'_y} \wedge newPred_{s',t'_y})\{\!\!\{Post_{OT} \uplus Post_{OT_y}\}\!\!\}\big)\big]\big\} \end{aligned} \quad (17)$$

And we know that Triple $(s, t|Pred_{s,t})$ has been updated before $\{Pred_{s',t'_y}\}^{y \in Y}$ was updated, and Triple $(s, t|Pred_{s,t})$ satisfies Definition 5.1 after last update. Thus, for any Triple predicate formula $oldPred_{weak}$ such that

$$\begin{aligned} oldPred_{weak} \implies \forall fv_{OT}.\big\{Pred_{OT} \Longrightarrow & \bigvee_{z \in X \setminus Y} [\exists fv_{OT_z}.(C_z \wedge Pred_{s',t'_z}\{\!\!\{Post_{OT} \uplus Post_{OT_z}\}\!\!\})] \\ & \bigvee_{y \in Y} \big[\exists fv_{OT_y}.\big(C_y \wedge oldPred_{s',t'_y}\{\!\!\{Post_{OT} \uplus Post_{OT_y}\}\!\!\}\big)\big]\big\} \end{aligned}$$

we know that

$$oldPred_{weak} \implies Pred_{s,t}$$

And according to basic rule of first order logic, we can derive following from formula 17:

$$\begin{aligned} Pred_{weak} \implies \forall fv_{OT}.\big\{Pred_{OT} \Longrightarrow & \bigvee_{z \in X \setminus Y} [\exists fv_{OT_z}.(C_z \wedge Pred_{s',t'_z}\{\!\!\{Post_{OT} \uplus Post_{OT_z}\}\!\!\})] \\ & \bigvee_{y \in Y} \big[\exists fv_{OT_y}.\big(C_y \wedge oldPred_{s',t'_y}\{\!\!\{Post_{OT} \uplus Post_{OT_y}\}\!\!\}\big)\big]\big\} \end{aligned}$$

thus, we know that

$$Pred_{weak} \implies Pred_{s,t}$$

and recall the structure of $newPred_{OT}$ in Formula 16, finally we can derive that

$$Pred_{weak} \implies newPred_{OT}$$

As the same reason we used during proving Beginning Triples, we proved that, after applying any Triple $(s, t|Pred_{s,t})$, which are not a Beginning Triple, to function *UpdatePredicate*, if we get the result $(s, t|newPred_{s,t})$, then Triple $(s, t|newPred_{s,t})$ satisfies the Definition 5.1.

Thus, in summary, we proved all the Triples in generated StrFH-Bisimulation satisfy the Defintion 5.1, thus Theorem 5.3 holds. □

D Check StrFH-Bisimilar

D.1 Proof of Algorithm Correctness

In this section, we will prove that the StrFH-Bisimilar checking algorithm is correct, which is formally expressed by Theorem 5.5.

Proof. Suppose we inputting two structural open automata A_1 and A_2 into the algorithm, if the result of algorithm is *True*, means for the Triple set $\mathcal{R}$ we generate by calling *GenerateWeakest-StrFH*, after deleting all unsatisfiable Triples in $\mathcal{R}$, the new Triple Set $\mathcal{R}_{refined}$ we got still contains intial Triple $(init_1, init_2|Pred_{init})$. And what we want to prove is: A_1 and A_2 are StrFH-Bisimilar.

As we mentioned previous section, function *GenerateWeakestStrFH* will generate a Triple Set $\mathcal{R}$, where $\mathcal{R}$ is definitely a StrFH-Bisimulation between A_1 and A_2.

Thus, the key point is to prove that $\mathcal{R}_{refined}$ is still a StrFH-Bisimulation between A_1 and A_2. If it is, $\mathcal{R}_{refined}$ is the StrFH-Bisimulation which meet the definition of StrFH-Bisimilar, means A_1 and A_2 are StrFH-Bisimilar.

Let Triple $(s, t|Pred_{s,t})$ be any Triple in $\mathcal{R}_{refined}$. For any open transition OT starting from s

$$OT = \frac{\beta_j^{j\in J'},\ Pred_{OT},\ Post_{OT}}{s \xrightarrow{\alpha} s'}$$

we can find a set of open transitions $\{OT_x\}^{x\in X}$, where for any open transition OT_x in this set, we have

$$OT_x = \frac{\beta_j^{j\in J'_x},\ Pred_{OT_x},\ Post_{OT_x}}{t \xrightarrow{\alpha_x} t'_x}$$

and there exist a corresponding Triple $(s', t'_x|Pred_{s',t'_x})$ in $\mathcal{R}$.

And through same method, we can find another set of open transitions $\{OT_y\}^{y\in Y}$, which for any open transition OT_y in this set, exist a Triple $(s', t'_y|Pred_{s',t'_y})$ in $\mathcal{R}_{refined}$. Obviously, $\{OT_y\}^{y\in Y} \subseteq \{OT_x\}^{x\in X}$

We can construct the following proof obligation po from OT and $\{OT_y\}^{y\in Y}$:

$$\begin{aligned}\forall fv_{OT}.\{Pred_{s,t} \wedge Pred_{OT} \Longrightarrow \\ \bigvee_{y\in Y} [\exists fv_{OT_y}.(\forall j \in J'.\beta_j = \beta_{j_y} \wedge \alpha = \alpha_y \\ \wedge Pred_{OT_y} \wedge Pred_{s',t'_y}\{\!|Post_{OT} \uplus Post_{OT_y}|\!\})]\}\end{aligned} \quad (18)$$

and we want to prove that the above formula is a tautology.

Due to the fact that $\mathcal{R}$ is a StrFH-Bisimulation between A_1 and A_2, we know that proof obligation generated from OT and set of open transitions $\{OT_x\}^{x\in X}$

$$\begin{aligned}\forall fv_{OT}.\{Pred_{s,t} \wedge Pred_{OT} \Longrightarrow \\ \bigvee_{x\in X} [\exists fv_{OT_x}.(\forall j \in J'.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_x \\ \wedge Pred_{OT_x} \wedge Pred_{s',t'_x}\{\!|Post_{OT} \uplus Post_{OT_x}|\!\})]\}\end{aligned}$$

is a tautology. And we can derive this formula to

$$\forall fv_{OT}.\Big\{Pred_{s,t} \wedge Pred_{OT} \Longrightarrow \bigvee_{y\in Y}\big[\exists fv_{OT_y}.(\forall j \in J'.\beta_j = \beta_{j_y} \wedge \alpha = \alpha_y \wedge Pred_{OT_y} \wedge Pred_{s',t'_y}\{\!\!\{Post_{OT} \uplus Post_{OT_y}\}\!\!\})\big] \vee \bigvee_{z\in X\setminus Y}\big[\exists fv_{OT_z}.(\forall j \in J'.\beta_j = \beta_{j_z} \wedge \alpha = \alpha_z \wedge Pred_{OT_z} \wedge Pred_{s',t'_z}\{\!\!\{Post_{OT} \uplus Post_{OT_z}\}\!\!\})\big]\Big\}$$

and we know that all the predicate formula $Pred_{s',t'_z}^{z\in X\setminus Y}$ are unsatisfiable, thus we can derive that:

$$\forall fv_{OT}.\Big\{Pred_{s,t} \wedge Pred_{OT} \Longrightarrow \bigvee_{y\in Y}\big[\exists fv_{OT_y}.(\forall j \in J'.\beta_j = \beta_{j_y} \wedge \alpha = \alpha_y \wedge Pred_{OT_y} \wedge Pred_{s',t'_y}\{\!\!\{Post_{OT} \uplus Post_{OT_y}\}\!\!\})\big] \vee \bigvee_{z\in X\setminus Y} False\Big\}$$

is a tautology. It means the formula 18 is a tautology.

Thus, we have prove that for any Triple in $\mathcal{R}_{refined}$, any proof obligation generated from it is always a tautology, means $\mathcal{R}_{refined}$ is also a StrFH-Bisimulation between A_1 and A_2. □

D.2 Proof of Algorithm Completeness

In this section, we will prove that the StrFH-Bisimilar checking algorithm is complete, which is formally expressed by Theorem 5.6.

Proof. Compared to proving theorem 5.6, we prefer to prove the Contrapositive of it. That is, for any two input open automata A_1 and A_2, as long as the problem is decidable by our algorithm and the output is $False$, then A_1 and A_2 are not StrFH-Bisimilar.

Suppose we input two open automata A_1 and A_2 into the algorithm, and states $init_1$ and $init_2$ are initial states of A_1 and A_2 respectively. If the result of algorithm is $False$, it means we generate a Triple set $\mathcal{R}$ by calling *GenerateWeakestStrFH* and in Triple $(init_1, init_2|Pred_{init}) \in \mathcal{R}$, the predicate formula $Pred_{init}$ is unsatisfiable. And what we want to prove is: A_1 and A_2 are not StrFH-Bisimilar.

Considering that if we suppose A_1 and A_2 are StrFH-Bisimilar, means there exist a StrFH-Bisimulation $\mathcal{R}_{weak}$ between A_1 and A_2, such that there exist a Triple $(init_1, init_2|Pred_{weak})$ in $\mathcal{R}_{weak}$ and $Pred_{weak}$ is satisfiable. We know that $Pred_{init}$ is unsatisfiable, thus $Pred_{weak}$ is weaker than $Pred_{init}$. But from Theorem 5.3, we know Triple set $\mathcal{R}_{weak}$ couldn't be a StrFH-Bisimulation between A_1 and A_2, contradictory with assumption.

Thus, we know that as long as the algorithm outputs $False$, then A_1 and A_2 are not StrFH-Bisimilar. □

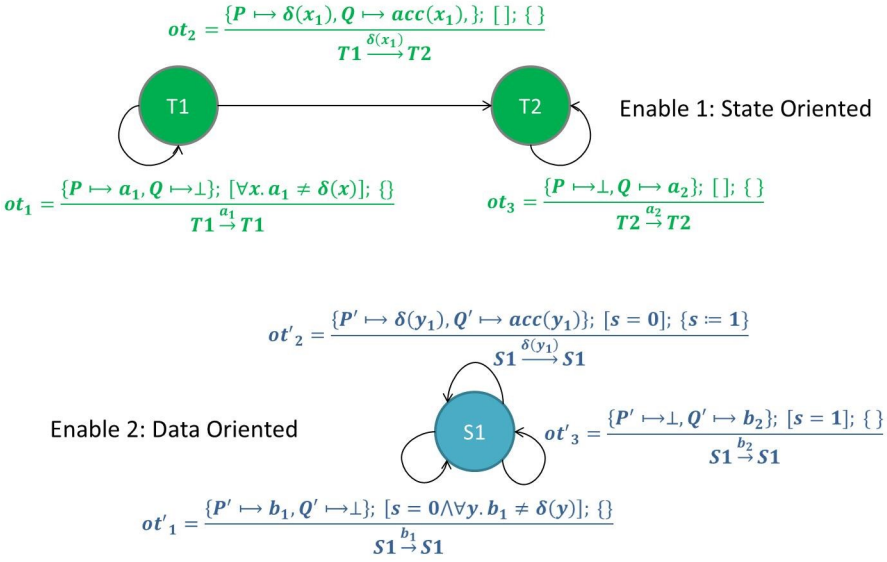


Figure 4: The two open automata

D.3 Running Example

In this section, we will provide the open automata in figure 4 above as a running example for the *Check StrFH-Bisimilar* algorithm.

Let us naming the first automaton as A_1 and second one as A_2. After accepting A_1 and A_2, algorithm will provide A_1 and A_2 as argument to function *GenerateWeakestStrFH* to generate the weakest StrFH-Bisimulation. Obviously, it will first call function *PushReachableStatePairs* and initialize the StatePair's stack(right side is top of stack):

$$StateStack = \sqsubset (T_1, S_1) \sqsubset (T_2, S_1)$$

and construct a initial Triple Set:

$$TripleSet = \{(T_1, S_1|True), (T_2, S_1|True)\}$$

Then, we pop the first element from *StateStack* and get StatePair (T_2, S_1), and call sub-function *UpdatePredicate* to update corresponding Triple $(T_2, S_1|True)$.

During updating, we first consider the open transition starting from state T_2. There are only one open transition from T_2, thus we will only generate one proof obligation from this side. Recall the form of proof obligation in definition is:

$$\begin{aligned}\forall fv_{OT}.\{Pred_{s,t} \wedge Pred_{OT} \implies \\ \bigvee_{x \in X} [\exists fv_{OT_x}.(\forall j \in J'.\beta_j = \beta_{j_x} \wedge \alpha = \alpha_{j_x} \\ \wedge Pred_{OT_x} \wedge Pred_{s',t'_x}\{\!|Post_{OT} \uplus Post_{OT_x}|\!\})]\}\end{aligned}$$

thus we will generate the following proof obligation

$$\begin{aligned}\forall a_2.\{True \wedge True \implies \\ False \bigvee False \bigvee [\exists b_2.(a_2 = b_2 \wedge a_2 = b_2 \wedge s = 1 \wedge True)]\}\end{aligned} \quad (19)$$

Noting that although there are three open transitions starting from S_1, but open transition ot'_1 and ot'_2 have different activated holes with ot_3, thus atomic formula $\forall j.\beta_j = \beta_{j_x}$ of them won't be satisfied, the corresponding formula are represented by $False$.

Moreover, the first atomic formula $a_2 = b_2$ is generated according to the atomic formula $\forall j.\beta_j = \beta_{j_x}$, and the second atomic formula $a_2 = b_2$ is generated according to the atomic formula $\alpha = \alpha_x$.

We know that formula 19 is not a tautology, thus we need to update predicate of Triple $(T_2, S_1|True)$ to a new predicate $Pred_{2,1}$. After removing the repeated atomic formula, we will have following:

$$Pred_{2,1} = \forall a_2.\{\exists b_2.(a_2 = b_2 \wedge s = 1)\}$$

and modifying name of bound variables to make it more readable and avoid conflict, we can have:

$$Pred_{2,1} = \forall a'.\{\exists b'.(a' = b' \wedge s = 1)\}$$

But for now, updating of Triple $(T_2, S_1|Pred_{2,1})$ is not finished. Consider the other side of Triple, there are three open transitions starting from S_1, means it will construct three proof obligation. First we construct the proof obligation for ot'_1:

$$\forall b_1.\{\forall a'.[\exists b'.(a' = b' \wedge s = 1)] \wedge s = 0 \wedge \forall y.[b_1 \neq \delta(y)] \implies False\}$$

It's easy to see, due to the fact that ot'_1 have a different activated hole with ot_3, the right side of "Implication" operator can be represented by $False$.

But the above formula is still a tautology, because the conflict of $s = 1$ and $s = 0$ on the left side of "Implication" operator, the above formula can be derived to $False \implies False$. And similarly, the proof obligation for ot'_2 also can be derived to $False \implies False$.

Finally consider the proof obligation constructed by ot'_3, we have:

$$\forall b_2.\Big\{\forall a'.[\exists b'.(a' = b' \wedge s = 1)] \wedge s = 1 \implies \exists a_2.\big\{b_2 = a_2 \wedge b_2 = a_2 \wedge True \wedge \forall a'.[\exists b'.(a' = b' \wedge s = 1)]\big\}\Big\}$$

which is obviously a tautology.

Thus, the updating for Triple $(T_2, S_1|Pred_{2,1})$ is finished. For this moment, we have StatePair's stack as following:

$$StateStack = \sqsubset (T_1, S_1)$$

and Triple Set as following:

$$TripleSet = \Big\{\Big(T_1, S_1|True\Big), \Big(T_2, S_1|\forall a'.\{\exists b'.(a' = b' \wedge s = 1)\}\Big)\Big\}$$

Due to the fact that Triple $(T_2, S_1|True)$ is updated to $(T_2, S_1|Pred_{2,1})$, we need to push the pre-StatePair of (T_2, S_1) into the stack. It's easy to see that both (T_2, S_1) and (T_1, S_1) are the pre-StatePair of (T_2, S_1), and (T_1, S_1) is already in the stack. So after pushing, the StatePair's stack is becoming following:

$$StateStack = \sqsubset (T_1, S_1) \sqsubset (T_2, S_1)$$

again.

Pop the top element in stack, we will get (T_2, S_1) again, thus we need to update Triple $(T_2, S_1|Pred_{2,1})$ one more time. It's easy to see that all the proof obligations of it are already tautology, thus the predicate of it won't be changed after this updating, and we don't to push any thing into stack this time.

Then we will pop top element once again, get StatePair (T_1, S_1), and call function *UpdatePredicate* to update Triple $(T_1, S_1|True)$.

There are two open transitions starting from T_1, thus we will need to construct two proof obligation from T_1 side. First we construct proof obligation from ot_1, and we will have following proof obligation:

$$\forall a_1.\{True \wedge \forall x.[a_1 \neq \delta(x)] \implies \exists b_1.\{a_1 = b_1 \wedge a_1 = b_1 \wedge s = 0 \wedge \forall y.[b_1 \neq \delta(y)] \wedge True\} \bigvee False \bigvee False\}$$

it's not a tautology, thus we will update predicate of Triple $(T_1, S_1|True)$ to $Pred_{1,1}$ where:

$$Pred_{1,1} = \forall a_1.\{\forall x.[a_1 \neq \delta(x)] \implies \exists b_1.\{a_1 = b_1 \wedge s = 0 \wedge \forall y.[b_1 \neq \delta(y)]\}\}$$

rename the bound variables, we will have:

$$Pred_{1,1} = \forall c'.\{\forall x'.[c' \neq \delta(x')] \implies \exists d'.\{c' = d' \wedge s = 0 \wedge \forall y'.[d' \neq \delta(y')]\}\}$$

And from open transition ot_2, we can also construct a proof obligation like following:

$$\forall x_1.\Big\{Pred_{1,1} \wedge True \implies False \bigvee \exists y_1.\{\delta(x_1) = \delta(y_1) \wedge acc(x_1) = acc(y_1) \wedge \delta(x_1) = \delta(y_1) \wedge s = 0 \wedge \forall a'.\{\exists b'.(a' = b' \wedge s = 1)\}\{\!\{(s \leftarrow 1)\}\!\}\} \bigvee False\Big\}$$

it is also not a tautology. According to the updating rule, the predicate of Triple $(T_1, S_1|Pred_{1,1})$ will be updated to:

$$\forall c'.\{\forall x'.[a_1 \neq \delta(x')] \implies \exists d'.\{c' = d' \wedge s = 0 \wedge \forall y'.[d' \neq \delta(y')]\}\} \bigwedge \forall x_1.\{True \implies \exists y_1.\{\delta(x_1) = \delta(y_1) \wedge s = 0\}\}$$

We've finished all the proof obligation from T_1 side. From the other side, there are three open transitions starting from S_1, and will construct three proof obligation. But all these three proof obligation are tautology, thus we won't show the detail of them.

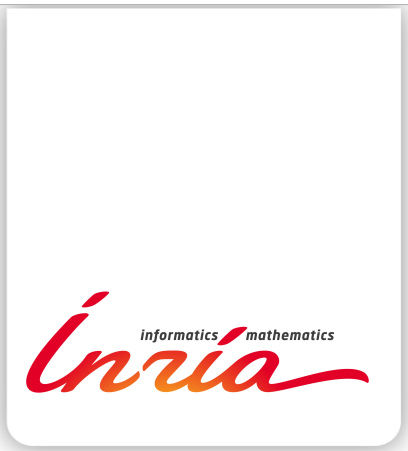
Finishing the updating of Triple $(T_1, S_1|True)$, and due to the fact that it's predicate has been changed, we need to push the pre-StatePair of (T_1, S_1) into stack. The only pre-StatePair of it is itself, thus we will have following situation after that:

$$StateStack = \sqsubset (T_1, S_1)$$
$$TripleSet = \{(T_1, S_1|Pred_{1,1}), (T_2, S_1|Pred_{2,1})\}$$

Pop the StatePair (T_1, S_1) again, and all the proof obligation for Triple $(T_1, S_1|Pred_{1,1})$ are tautology. Thus the *StateStack* become empty, and the function terminate, we have the Triple Set following:

$$
\begin{aligned}
TripleSet = \Big\{ & \\
& \Big(T_1, S_1 \mid \forall c'.\big\{\forall x'.[a_1 \neq \delta(x')] \implies \\
& \exists d'.\{c' = d' \wedge s = 0 \wedge \forall y'.[d' \neq \delta(y')]\}\big\} \\
& \bigwedge \forall x_1.\big\{True \implies \exists y_1.\{\delta(x_1) = \delta(y_1) \wedge s = 0\}\big\}\Big) \\
& \qquad , \Big(T_2, S_1 \mid \forall a'.\{\exists b'.(a' = b' \wedge s = 1)\}\Big)\Big\}
\end{aligned}
$$

we know that Triple Set is the weakest StrFH-Bisimulation between A_1 and A_2, and both the predicate formula in two Triple are satisfiable. Thus it's easy to see that A_1 and A_2 are StrFH-Bisimilar.



RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399